Relatório do Projeto — Mineração de Dados

Carlo Domenico Longo de Lemos, Daniel de Jesus Siqueira

1 Introdução

Com o objetivo de maximizar o lucro da concessionária, analisamos os dados de negociações realizadas em diferentes lojas da empresa. Aqui temos um total de 10 mil visitas realizadas entre janeiro de 2017 e setembro de 2019. Em cada linha temos atributos relacionados ao cliente, dados dos carros vistos e informações sobre os possíveis carros comprados.

A análise dos dados foi realizada em Python com o auxílio das bibliotecas pandas (para a criação de *dataframes* que facilita o tratamento dos dados), numpy (que oferece algumas funções matemáticas úteis), matplotlib (para criação de gráficos e visualização dos dados) e sklearn (que possui classificadores, regressores e métricas de validação já implementados).

A etapa de análise exploratória revelou que a proporção de negociações sucedidas (que resultaram em venda) nas lojas 3 e 4 era maior que nas lojas 1 e 2. Investigando o fenômeno, procuramos variáveis importantes que estavam distribuídas de forma desproporcional nas lojas. Isso nos motiva a criar campanhas para grupo de clientes específicos.

Além disso, utilizamos um classificador Random Forest para tentar prever, dado um cliente e um carro disponível da loja, se o cliente irá comprar esse carro. Usando esse modelo, a cada cliente novo, podemos oferecer os carros com os maiores valores de "lucro esperado" (definido pela probabilidade do cliente comprar esse carro multiplicada pelo log do valor do carro). Desta forma, maximizamos o lucro da empresa (assumindo que o rendimento da venda de um carro seja proporcional ao seu valor). Com isso, fixado um cliente que visita a loja, criamos, a partir da base de dados dos carros disponíveis na loja da concessionária, um mecanismo para escolher qual carro oferecer.

2 Análise Exploratória

O primeiro passo da nossa análise exploratória foi criar um *dataframe* do pandas com a base de dados fornecida, de forma a facilitar a adequação dos dados aos algoritmos e métodos de visualização que iremos utilizar.

Primeiro notamos que alguns atributos ['gesticula_negociante', 'automatico_primeiro_carro', 'flex_primeiro_carro', 'automatico_segundo_carro', 'flex_segundo_carro', 'automatico_carro_comprado', 'flex_carro_comprado', 'tem_criancas'] assumiam valores do tipo boolean. Como os algoritmos clássicos do sklearn não lidam bem com variáveis desse tipo, foi realizada uma conversão simples de boolean para float levando o False em 0 e o True em 1 (idealmente iríamos mudar para int, mas o int não aceita atributos faltantes). De forma similar, o atributo 'id_vendedor' assumia valores do tipo string. A solução foi uma conversão simples de parte da string que nos dava o número do vendedor para int.

Foi possível identificar dois atributos ordinais ['idade_negociante', 'vestimenta_negociante'] que assumiam valores do tipo string. Foi feita, então, uma conversão para int, preservando a ordem lógica identificada em cada atributo. Esse passo é importante, pois alguns algoritmos se utilizam da distância entre os atributos. Essa distância seria ignorada se tratássemos esses atributos como nominais.

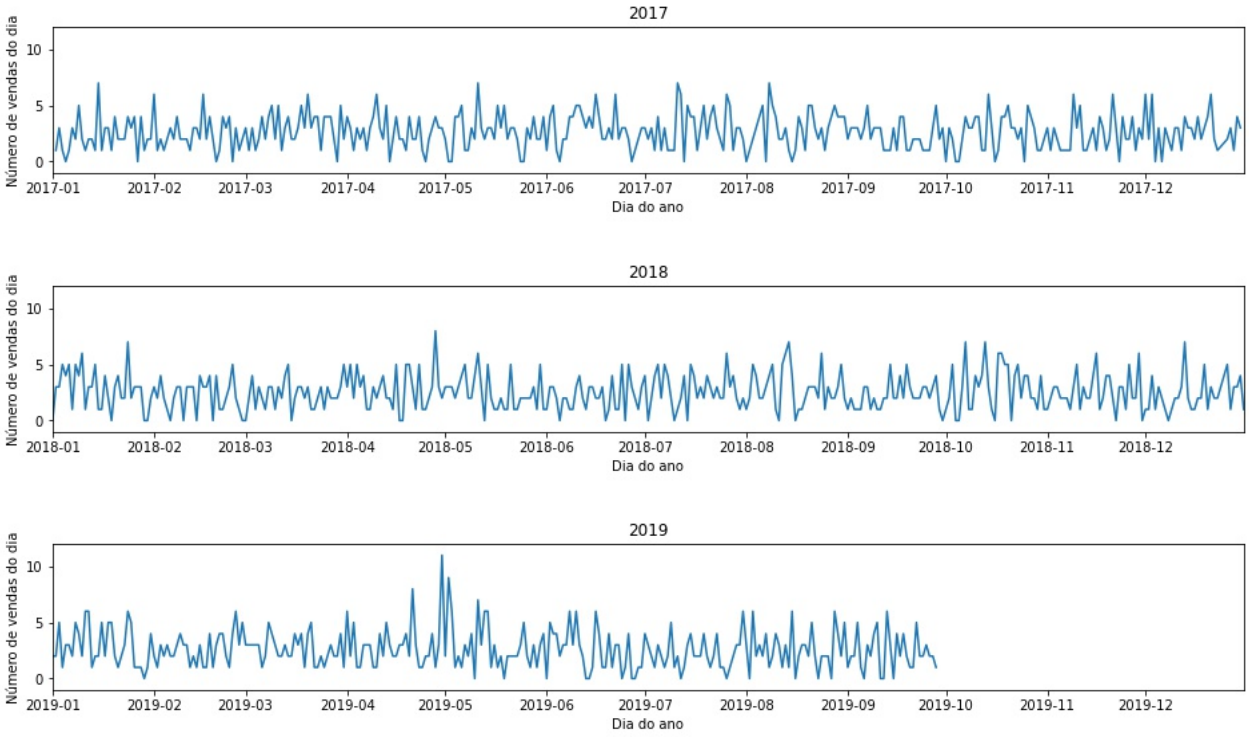
Com o objetivo de extrair mais informações, decidimos criar alguns atributos úteis. A flag 'comprou', que indica se uma visita resultou em venda ou não pode ser facilmente obtida verificando se o atributo 'valor_carro_comprado' é do tipo NaN ou não. Além disso, achamos interessante transformar o atributo 'data' em um objeto datetime. Feito isto, adicionamos os atributos ['dia_semana', 'dia_mês', 'mês', 'ano'] que são obtidos por meio de métodos do objeto datetime.

Agora, para os atributos nominais ['periodo_visita', 'sexo_negociante', 'cor_cabelo_negociante', 'cor_primeiro_carro', 'cor_segundo_carro', 'cor_carro_comprado', 'tipo_primeiro_carro', 'tipo_segundo_carro', 'tipo_carro_comprado', 'marca_primeiro_carro', 'marca_segundo_carro', 'marca_carro_comprado'], podemos atribuir um int para cada categoria, ou podemos criar novos atributos binários, com o get_dummies, para cada categoria de modo a não introduzir uma noção de distância errônea nesses atributos.

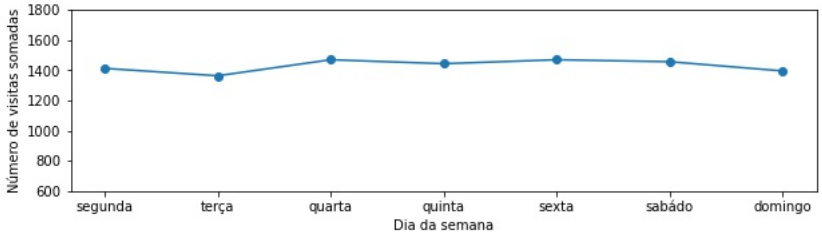
2.1 Análise Temporal

Decidimos plotar alguns gráficos que relacionavam a quantidade de vendas com o período do ano, com o objetivo de encontrar alguma relação. Os gráficos de vendas por dia para cada ano são os seguintes.

Notamos que, com alguns dias atípicos, o número de vendas flutua aleatoriamente em torno de 2 vendas por dia. Desta forma, os gráficos não geraram nenhum *insight* que relaciona o período do ano ou do mês com o número de vendas. Mas nos deixam confortáveis para assumir que a distribuição de vendas parece se manter constante no tempo. Isso é importante para garantir que o nosso modelo supervisionado, treinado com os dados do passado, se manterá eficiente e performando bem nos dados do futuro.



Verificamos ainda se o dia da semana, dia do mês e mês do ano nos fornecia algum padrão que pudessemos explorar. Plotando o gráfico que relaciona o dia da semana com as visitas, não é possível concluir que há alguma relação clara. O mesmo acontece nas outras variáveis.



2.2 Inconsistências

No processo de exploração, tentamos encontrar relações entre alguns atributos usando tabelas pivô. Uma delas apresentou uma inconsistência entre dois atributos importantes: 'nro_pessoas' (que indica o tamanho do grupo que acompanha o negociante) e a flag 'tem_criancas' (que indica se o negociante veio acompanhado de criança ou não):

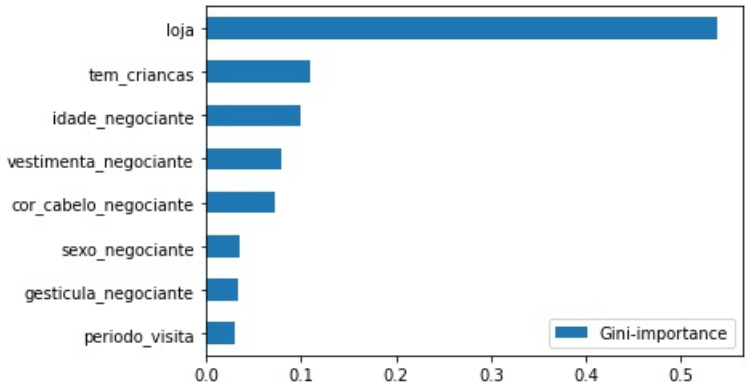
nro_pessoas	Não tem crianças	Tem crianças
1	248	5559
2	1995	123
3	822	54
4	457	31
5	671	40

Nota-se que em 5559 amostras (do total de 10000 visitas), o cliente veio sozinho, mas estava acompanhado de uma criança. Optamos, então, por remover o atributo 'nro_pessoas' de análises futuras, dado que a flag 'tem_criancas' é menos suscetível a erros por ser um atributo binário simples.

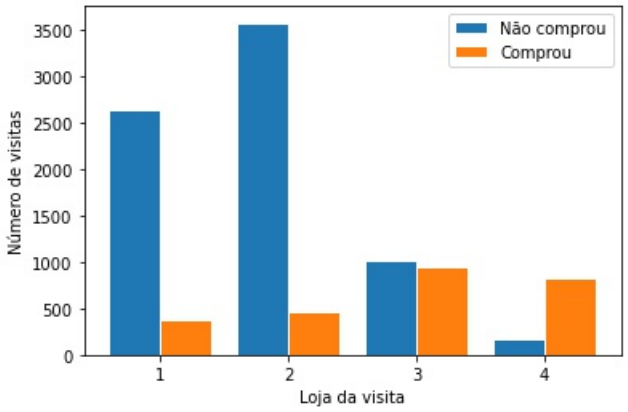
2.3 Atributos do cliente

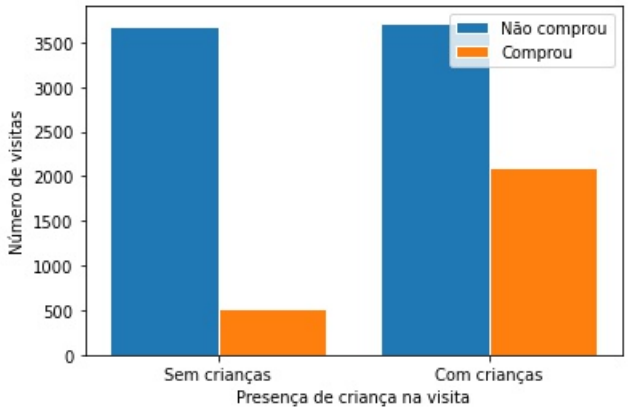
Agora, analisamos o perfil dos clientes, tentando identificar padrões que possam indicar se um novo cliente tem uma boa chance de fazer uma compra ou não. Para isto, vamos usar um classificador Random Forest que recebe os atributos do cliente e tenta prever o valor da flag 'comprou'. Analisamos superficialmente algumas métricas de classificação e, em seguida, identificamos quais atributos possuem maior importância para a classificação. Aqui, consideramos que ['loja', 'vestimenta_negociante', 'sexo_negociante', 'cor_cabelo_negociante', 'gesticula_negociante', 'idade_negociante', 'tem_criancas', 'periodo_visita'] são os atributos do cliente.

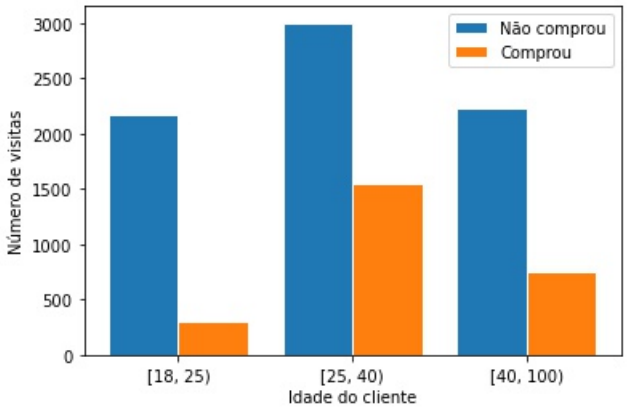
Sem fazer uma otimização dos hiperparâmetros e apenas delimitando que a profundidade máxima das árvores criadas seja 20, o modelo performa de forma razoável, obtendo um valor de ROC-AUC de 82% em um conjunto de teste separado com o train_test_split. Isso nos permite crer que as variáveis selecionadas nas árvores são importantes para prever a nossa target e nos motiva a olhar com mais cuidado para os atributos com maior importância:

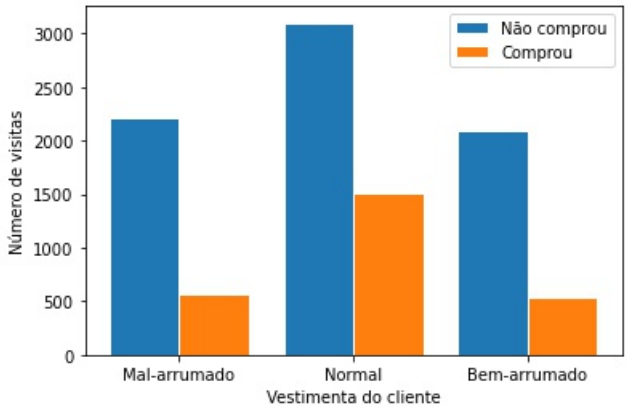


Com isso, identificamos que os atributos ['loja', 'tem_criancas', 'idade_negociante', 'vestimenta_negociante'] são os mais importantes. Plotando os gráficos de como os valores dos atributos mais importantes dividem o conjunto de dados em visitas que efetivamente viraram uma venda e visitas que não foram eficientes, temos:





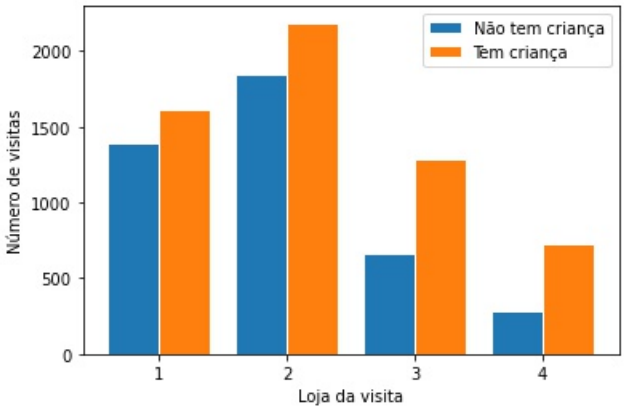




A loja é um atributo essencial para concluir se uma visita vira uma compra ou não. As lojas 1 e 2 tem um desempenho muito menor que o das lojas 3 e 4. Tentamos cruzar a informação da loja com informações dos clientes. Essa investigação nos demonstrou a relação com o atributo 'tem_criancas':

Ou seja, a proporção de clientes que visitam a loja com crianças é maior justamente nas lojas 3 e 4. Desta forma, se conseguirmos aumentar a proporção de clientes que visitam com crianças nas lojas 1 e 2, é possível que ocorra um aumento nas vendas.

Isso nos motiva a criar nossa primeira sugestão de intervenção. Uma possível estratégia para aumentar a eficiência das lojas 1 e 2 é incentivar os clientes a visitarem a loja com a família, através da realização de eventos com atrações que cativam as crianças ou até mesmo de minicursos interativos sobre trânsito.



3 Metodologia

Assumindo que atualmente os carros são oferecidos de maneira aproximadamente aleatória para os clientes, vamos criar um classificador que dada as variáveis do negociante e as variáveis do carro tentará prever se o cliente comprará aquele carro ou não. Com esse modelo, dado os carros disponíveis em estoque, iremos oferecer para o cliente aquele carro que tenha a melhor chance de ser vendido ponderado pelo log do valor do carro. A motivação para utilização do log baseia-se em um problema da teoria da utilidade, que nos diz que a função de utilidade monetária se comporta como log. A intuição dessa transformação é que a escala logarítmica nos da uma proporção mais adequada para lidar com riscos. Neste cenário, estamos mais confortáveis para aceitar riscos maiores apenas se o retorno esperado for proporcionalmente maior. Desta forma, levamos em conta o risco de oferecer um carro mais caro, mas com probabilidade menor de ser comprado.

Essa ideia de *Uplift modelling* nos permite escolher, qual o melhor tratamento entre os disponíveis (os carros no estoque de uma determinada loja) para um determinado cliente visando maximizar uma métrica de resultado.

Para construir esse classificador precisamos adaptar nossa base de dados criando novos exemplos a partir dos carros vistos em uma visita.

3.1 Criação das instâncias

Queremos treinar um classificador que consiga relacionar os atributos de um cliente fixo com os atributos de um carro disponível para venda. Uma observação importante nos permite combinar esses atributos de forma que o número de amostras aumente e o número de atributos por amostra diminua. Se criarmos uma nova flag 'olhou_segundo' que indica se um cliente olhou mais de um carro, temos que sua relação com a flag 'comprou' é:

	Não comprou	Comprou
Olhou só um	2453	847
Olhou mais de um	4948	1752

Agora, normalizando pela soma na linha obtemos:

	Não comprou	Comprou
Olhou só um	0.743333	0.256667
Olhou mais de um	0.738507	0.261493

Ou seja, a quantidade de carros vistos não influencia na decisão do cliente. Com isso em mente, dado um cliente do conjunto de teste, podemos criar mais amostras da forma ['valor', 'ano', 'km', 'automatico', 'flex', 'cor', 'marca', 'tipo', 'tem_criancas', 'vestimenta_negociante', 'idade_negociante', 'loja', 'y'], onde 'y' é uma flag que indica se o cliente comprou esse carro em específico (diferente de 'comprou', que apenas indicava se um cliente fez uma compra ou não) olhando para cada um dos carros vistos naquela visita.

Por exemplo, um cliente que olha 2 carros e compra o segundo, vai ser transformado em duas linhas dessa nova base. Em uma das linhas teremos seus atributos (que filtramos apenas para ser os relevantes da análise feita na parte exploratória) e as características do primeiro carro visto, com $y = 0$, pois esse carro não foi comprado. Já na outra linha teremos as variáveis daquele mesmo cliente, os atributos do segundo carro visto e $y = 1$ já que este foi um carro comprado. Outros exemplos seguem a lógica de forma análoga.

Desta forma, espera-se que o modelo consiga perceber melhor as preferências de carro para um dado perfil de cliente.

3.2 Comitês: Random Forest Classifier

Em linhas gerais, os comitês ou *ensembles* são uniões de modelos (normalmente mais simples, neste caso, chamados de *weak learners*) em um modelo mais complexo. Cada um desses modelos individuais nos dá uma previsão e isso é

sumarizado em um score (como por exemplo a proporção de votos em cada uma das classes).

Nas Random Forests, utilizamos árvores de decisão como *weak learners*. Cada uma das árvores é treinada em uma amostra por bootstrap da nossa base de treinamento. A amostra por bootstrap é uma amostragem com repetição. Ela nos garante que a distribuição dos dados amostrados é a mesma que a do conjunto inicial, sem precisar pegar todos os mesmos elementos que tinhamos inicialmente. Essa forma de seleção de exemplos nos garante que cada uma das árvores vai aprender com instâncias um pouco diferentes em seu treinamento, mas que tem a distribuição aproximadamente parecida com os dados de treino. As amostras não selecionadas para treinar uma determinada árvore é o conjunto *out of bag*. Podemos inclusive avaliar o desempenho individual daquele classificador nesse conjunto que não foi utilizado para treiná-lo.

Além disso, para garantir que essas árvores sejam o mais independentes possível, é forçado que, no treinamento de cada uma, cada quebra seja feita olhando apenas um subconjunto aleatório de atributos. Desta forma, olhamos o ganho de informação naquele subconjunto de características e a variável escolhida é a que maximiza o ganho de informação entre elas.

Essa construção nos dá árvores individualmente piores, mas a garantia de termos erros mais independentes é boa pois a chance de muitas árvores errarem um mesmo exemplo é menor. Não queremos que os *weak learners* sejam muito robustos. Apenas que eles classifiquem de forma razoável e preferencialmente errando exemplos em regiões diferentes.

Alguns hiper-parâmetros relevantes que utilizamos são o número de árvores, a profundidade máxima e o número mínimo de exemplos em cada folha.

Uma coisa importante que podemos calcular com as Random Forest Classifiers é uma estimativa da probabilidade de ser de uma determinada classe. Para cada uma das árvores podemos definir a probabilidade de ser de uma determinada classe como sendo a fração de exemplos daquela folha que são daquela classe. A Random Forest sumariza essas quantidades falando que a probabilidade de um exemplo ser de uma determinada classe é a média das probabilidades que cada uma das árvores nos dá.

3.2.1 Feature Importance

As Random Forests são classificadores muito utilizados por normalmente apresentarem métricas de erro razoáveis e serem rápidas de aplicar (estamos apenas evaluando um exemplo no número de árvores escolhido anteriormente). Mas uma outra vantagem relevante é que ela nos oferece um pouco de interpretabilidade também.

As implementações clássicas das Random Forests, costumam apresentar um cálculo de importância das variáveis. Usualmente é utilizado o fator de importância gini. Nele, para cada uma das variáveis, somamos o ganho de informação gini que cada uma das quebras daquela variável nos deu, em cada uma das árvores. Dessa forma, se temos variáveis que contrubuíram bastante para criar nós mais puros, o ganho de informação delas é alto e, portanto, apresentam uma *feature importance* alta.

Essa é uma maneira razoável de entender quais os atributos estão sendo mais relevantes nos treinamentos das árvores. Utilizamos essa validação para guiar nossa análise exploratória anteriormente.

3.3 Métrica de classificação: ROC-AUC

A curva ROC nos dá, no eixo x a taxa de falsos positivos, enquanto no eixo y temos a taxa de verdadeiros positivos quando variamos o limiar da probabilidade de ser de uma determinada classe que são retornadas pelo Random Forest Classifier.

Ela nos dá uma medida de quão bem ordenados nossos scores de pertencimento às classes estão. Essas curvas sempre passam nos pontos (1,1) (no threshold em que todos os exemplos são classificados como da classe positiva) e (0,0) (no threshold em que todos os exemplos são classificados como da classe negativa). Num classificador ideal, que realmente consegue ordenar os dados de forma que existe um limiar que separa as duas classes, é possível atingir o ponto (0,1) em que não há falsos positivos e estamos acertando todos os positivos. Dessa forma, a curva idealizada tem como região máxima o quadrado $[1, 1] \times [1, 1]$.

Dada essa motivação, uma forma de sumarizar a curva ROC é calculando sua área debaixo da curva entendendo que esse valor é ideal quando vale 1. Esse valor, chamado de ROC-AUC, pode ser interpretado como a probabilidade de que um objeto positivo seja ranqueado acima de um objeto negativo. Esse tipo de métrica nos parece apropriada pois queremos ver quais os carros que têm maior chance de serem comprados e não necessariamente se serão comprados ou não.

3.4 Desbalanceamento das classes

Depois de criar nossos dados como descrito na seção 3.1, ficamos com um problema de classificação muito desbalanceado. Dos 17594 exemplos, apenas 14,7% são instâncias onde há compra. A maioria dos métodos tem problemas para trabalhar nesse tipo de cenário e vimos em aula algumas estratégias de abordar esse problema, como dando pesos diferentes para os erros cometidos ou reamostrando nossos dados de uma forma que o problema fique balanceado (repetindo dados da classe minoritária ou excluindo exemplos da classe maioritária).

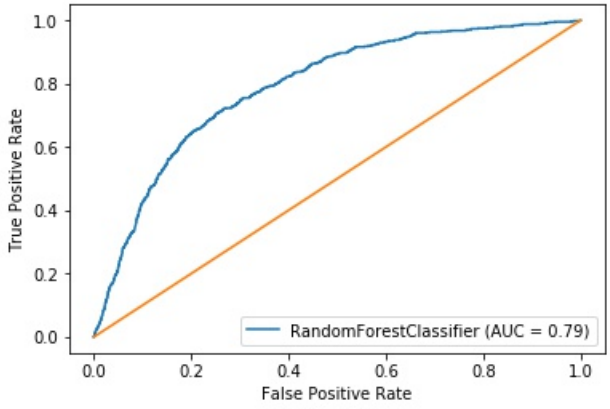
Aqui, decidimos utilizar um método específico do contexto de algoritmos de bagging, o *balanced_subsample*. Nessa abordagem, garantimos que em todas as amostragens por bootstrap da criação das árvores tenham uma proporção

balanceada de exemplos de cada uma das classes. Dessa forma, as árvores que constituem nossa floresta não ficam tendenciosas a votar na classe majoritária.

4 Resultados

Dividimos nossa base em dois conjuntos, um para treino e outro para teste. Fizemos um grid_search otimizando alguns hiper-parâmetros com validação cruzada dos dados de treinamento para a Random Forest. Estamos vendo todas as combinações de profundidade máxima sendo 3, 13, 23, 33 ou 43, mínimo de exemplos por folha sendo 10, 90, 170 ou 250 e número de árvores como 100, 200 ou 300.

Os melhores parâmetros são então fitados em todos os conjuntos de treinamento para vermos seu desempenho no conjunto de teste. Obtemos um ROC-AUC de 0.79, com curva ROC dada por:



Isso nos permite acreditar que o ranqueamento é razoável e que o modelo está aprendendo um padrão apesar de não ser muito poderoso.

4.1 Funcionamento do modelo na prática

Para exemplificar nossas análises, vamos mostrar como isso funcionaria na prática. Imagine que estamos em alguma das lojas, por exemplo, a loja 2. Com algum estoque de carros (eles foram amostrados da nossa tabela original no subconjunto de instâncias com loja = 2) dado pela tabela a seguir:

Índice do carro	Valor	Ano	Km	Automático	Flex	Cor	Marca	Tipo
1	72057	2012	157	False	True	Branco	Renault	Sedan
2	59040	2012	112	False	False	Vermelho	Renault	SUV
3	47603	2016	140	True	True	Vermelho	Renault	Hatch
4	74740	2006	186	True	False	Branco	Fiat	Hatch

Agora, um cliente chega na loja 2 com as variáveis:

Tem criança	Vestimenta	Idade
True	Normal	[40, 100)

Para cada um dos carros, fixamos as variáveis do cliente, criando tabela de input para o modelo. O modelo calcula então a probabilidade de cada carro ser comprado por aquele cliente. Isso é ponderado pelo log do valor do carro, como vemos na tabela:

Índice do carro	Probabilidade de ser vendido	Valor do carro	Probabilidade vezes $\log_2$ do preço
1	0.47853016	72057	7.721975
2	0.45953833	59040	7.283409
3	0.47691476	47603	7.410673
4	0.46395431	74740	7.511231

Essa tabela nos indica um ranking de sugestão de venda, guiando a estratégia de oferecimento de carros do vendedor. Ele pode ordenar pelo score, nos indicando que é melhor vender o carro de índice 1, mesmo que este não seja o carro mais caro.

Como segunda opção, o vendedor pode escolher oferecer o carro que tem maior probabilidade de venda entre os carros disponíveis. Utilizando o nossos resultados exclusivamente como um modelo de propensão. No caso exemplificado, ofereceríamos o carro de índice 3, levando em conta que já tentamos oferecer o índice 1 antes.

4.2 Sanity Check

Podemos achar que o valor do carro pode influenciar demais no nosso score criado, ou que a probabilidade estaria dominando por causa da aplicação do log. Este não parece ser o caso aqui. Fizemos um pequeno código para simular a aplicação desse algoritmo na prática.

O algoritmo simula a ida de vários clientes nas lojas com estoques diferentes. A ideia é que as variáveis do cliente e da loja sejam sorteadas de maneira uniforme entre as possíveis e daí tiramos 20 carros amostrados dos dados originais. Aplicamos nosso algoritmo para indicar o carro que maximiza a probabilidade vezes o log do valor e vemos se esse carro é o que maximiza somente as probabilidade, somente os valores ou dois ao mesmo tempo.

Das 1000 simulações, em 454 das vezes, o carro selecionado era um que maximizava a probabilidade, mas não era o de maior valor. Já em 138 vezes o valor foi predominante, e escolhemos o carro com maior valor, mesmo ele não sendo que tinha a maior probabilidade. Em 158 vezes estavámos em um cenário ideal em que o carro com maior valor era também o carro com maior probabilidade de ser comprado. Em 252 vezes, o carro escolhido não era nem o mais caro, nem o com maior probabilidade, o que também é esperado de acontecer as vezes.

Essa validação nos permite confiar mais no algoritmo, uma vez que ele tem uma lógica de negócio baseada na teoria da utilidade (em que podemos preferir garantir lucros menores, mas mais garantidos a ousar lucros altos com risco maior). Ainda sim, não estamos utilizando-o apenas como um modelo de propensão. Incorporamos uma métrica de lucro (traduzida pelo valor) de forma a otimizar simultaneamente uma oferta mais acertiva e com maior retorno financeiro.

5 Comentários Finais

Apesar dos resultados coerentes, não estamos totalmente confortáveis com a métrica que estamos tentando otimizar. Idealmente, tendo um valor de probabilidade mais fiel, poderíamos usar o próprio valor esperado da venda, dado por valor do carro vezes a probabilidade de ele ser vendido. Neste caso, as probabilidades dariam valores mais distantes, e não precisaríamos fazer a transformação logarítmica motivada pela teoria da utilidade.

Uma forma que vimos para tentar lidar com essas probabilidade foi a postagem sobre calibração de probabilidades do sklearn. Em que, em linhas gerais, aplica-se uma regressão logística nas saídas. Entretanto, isso não pareceu ajudar muito e contínuamos com valores muito próximos. Outra maneira de lidar poderia ser a aplicação de regressões logísticas nos exemplos de folhas de cada árvore do nosso comitê. Essa metodologia foi vista em um dos links na referência, mas não foi testada aqui.

O desbalanceamento das classes também foi um grande problema, a abordagem utilizando o *balanced_subsample* foi a melhor entre as testadas, mas não fica claro se poderíamos ter escolhido uma técnica mais acertiva.

Inicialmente, gostaríamos de segmentar os clientes, mas era díficil escolher uma distância apropriada pelas variáveis serem categóricas e não era claro que tipo de intervenção poderíamos propor a partir dos grupos encontrados.

Concluindo, cremos que os tratamentos sugeridos merecem atenção e devem ser utilizados de forma controloda com testes A/B para medir suas eficiências. A sugestão é aplicar os eventos familiares em uma das lojas de desempenho menor (1 ou 2) e avaliar se seu desempenho se destaca em relação à outra. Enquanto no caso do modelo, sugerimos o teste com alguns dos funcionários uma vez que o desempenho de vendas individual não se mostrou ser relevante nas análises exploratórias. Se esses funcionários começarem a ter um número de vendas maior do que os outros, podemos concluir que o modelo cumpre o seu papel.

6 Refêrencias Extra Aulas

- Uplift modeling: https://en.wikipedia.org/wiki/Uplift_modelling
- Feature importance: https://towardsdatascience.com/the-3-ways-to-compute-feature-importance-in-the-random-forest-96c86b49e6d4
- Risk aversion: https://en.wikipedia.org/wiki/Risk_aversion
- Probability calibration: https://scikit-learn.org/stable/modules/calibration.html
- Calibration of probabilities for tree-based models: https://gdmarmerola.github.io/probability-calibration/

7 Vídeo da apresentação

https://drive.google.com/file/d/1ZpU-ZFpwnELWjG-44KGwUb9ibw5CRvMp/view?usp=sharing