Nanodegree Engenheiro de Machine Learning

Projeto final

Alexandre Ray da Silva

09/08/2019

I. Definição

Visão geral do projeto

O sistema de compartilhamento de meios de transporte é uma das alternativas de mobilidade urbana que vem ganhando cada vez mais espaço em grandes cidades como São Paulo. Mais conhecido como MaaS (Mobilidade-como-um-Serviço) representa uma mudança de propriedade pessoal de meios de transporte para soluções de mobilidade que são consumidos como um serviço.

No caso dos serviços de compartilhamento de bicicletas (bike sharing), elas são disponibilizadas para que qualquer pessoa possa utilizá-la por meio do desbloqueio a partir de um aplicativo de celular. Em geral, é permitido que o usuário alugue a bicicleta em um local e a deixe em outro lugar ou estação diferente da inicial conforme a necessidade de locomoção, proporcionando maior flexibilidade para os usuários. Além disso, a rede de sensores das bicicletas fornecem uma rica quantidade de dados que podem ser usados para estudo de mobilidade nas cidades.

Descrição do problema

O problema consiste em predizer a demanda pelo aluguel de bicicletas usando o histórico de demanda de bicicletas combinados com dados climáticos para o programa da Capital Bikeshare em Washington, D.C. A predição terá como referência o intervalo de uma hora. Dessa forma, queremos responder perguntas como:

1. Quantas bicicletas serão alugadas no dia 20/07/2019 às 15 horas?
2. E às 16 horas?

O conhecimento prévio da demanda pelo aluguel de bicicletas pode endereçar melhores estratégias de distribuição desse meio de transporte pela cidade, proporcionando melhores experiências para os usuários desses serviços.

Métricas

A métrica de avaliação a ser utilizada nesse projeto é a Root Mean Squared Logarithmic Error (RMSLE). A métrica RMSLE geralmente é usada quando não queremos penalizar muito as diferença entre o valor real e o valor predito no caso em que o valor real e predito são muito grandes.

Esse modo de lidar com valores muito grandes é interessante para o problema aqui proposto. Por exemplo, quando a demanda de bicicletas atinge um valor considerado muito alto, o tamanho do erro passa a não importar mais. Em outras palavras, a demanda de 10k bicicletas em uma determinada região já não é tão diferente de 100K bicicletas uma vez que a situação já pode ser considerada crítica (altíssima demanda) na primeira situação. Matematicamente a métrica RMSLE é descrita como:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

II. Análise

Exploração dos dados

A tabela abaixo mostra todas as variáveis que foram disponibilizadas para esse projeto, bem como uma breve descrição e o tipo:

Variável	Descrição	Tipo
datetime	hourly date + timestamp	Data
season	1 = spring, 2 = summer, 3 = fall, 4 = winter	Categórica
holiday	whether the day is considered a holiday	Binária
workingday	whether the day is neither a weekend nor holiday	Binária
weather	1: Clear, Few clouds, Partly cloudy, Partly cloudy	Categórica

	2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog	
temp	temperature in Celsius	Numérica
atemp	"feels like" temperature in Celsius	Numérica
humidity	relative humidity	Numérica
windspeed	wind speed	Numérica
casual	number of non-registered user rentals initiated	Numérica
registered	number of registered user rentals initiated	Numérica
count	number of total rentals	Numérica

Segue uma amostra dos dados usando a função *head* do pandas:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

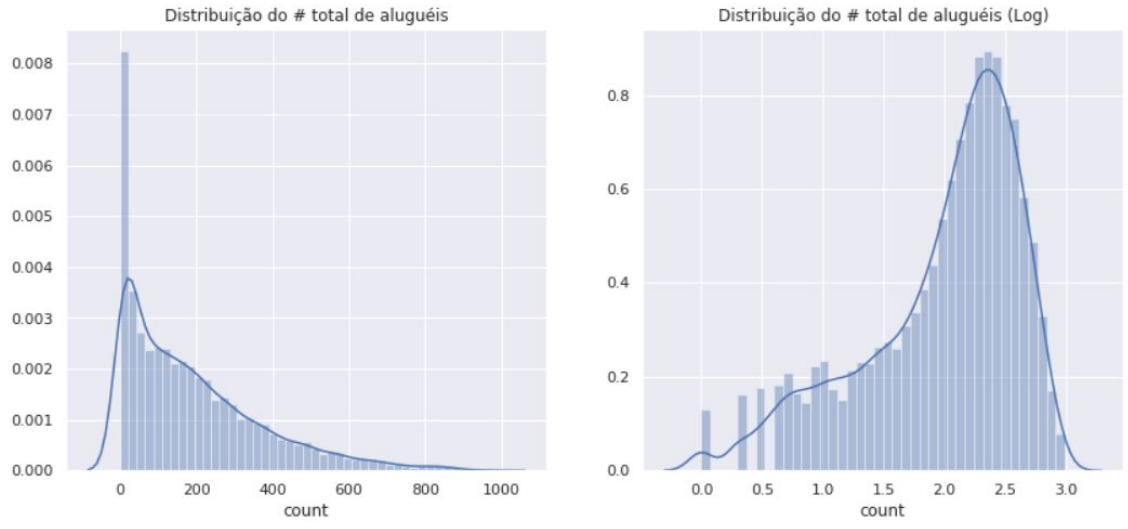
Para esse conjunto de dados, não foram encontrados valores ausentes. Na tabela abaixo, podemos ver detalhadamente a distribuição dos dados de cada uma das variáveis do problema:

	count	mean	std	min	25%	50%	75%	max
season	10886.0	2.506614	1.116174	1.00	2.0000	3.000	4.0000	4.0000
holiday	10886.0	0.028569	0.166599	0.00	0.0000	0.000	0.0000	1.0000
workingday	10886.0	0.680875	0.466159	0.00	0.0000	1.000	1.0000	1.0000
weather	10886.0	1.418427	0.633839	1.00	1.0000	1.000	2.0000	4.0000
temp	10886.0	20.230860	7.791590	0.82	13.9400	20.500	26.2400	41.0000
atemp	10886.0	23.655084	8.474601	0.76	16.6650	24.240	31.0600	45.4550
humidity	10886.0	61.886460	19.245033	0.00	47.0000	62.000	77.0000	100.0000
windspeed	10886.0	12.799395	8.164537	0.00	7.0015	12.998	16.9979	56.9969
casual	10886.0	36.021955	49.960477	0.00	4.0000	17.000	49.0000	367.0000
registered	10886.0	155.552177	151.039033	0.00	36.0000	118.000	222.0000	886.0000
count	10886.0	191.574132	181.144454	1.00	42.0000	145.000	284.0000	977.0000

Podemos observar pela variável que queremos prever (count) que sua média é ~191 bicicletas alugadas e seu valor máximo é de 977. No dia mais quente dessa amostra de dados a temperatura foi de 41 graus Celcius e a sensação térmica foi de 45 graus Celcius. A umidade possui escala de 0 a 100 e a velocidade do vento pode ir até 56km/h. Para as variáveis numéricas, parece não haver valores muito estranhos. Usando visualização de dados, vamos explorar mais a frente as variáveis categóricas.

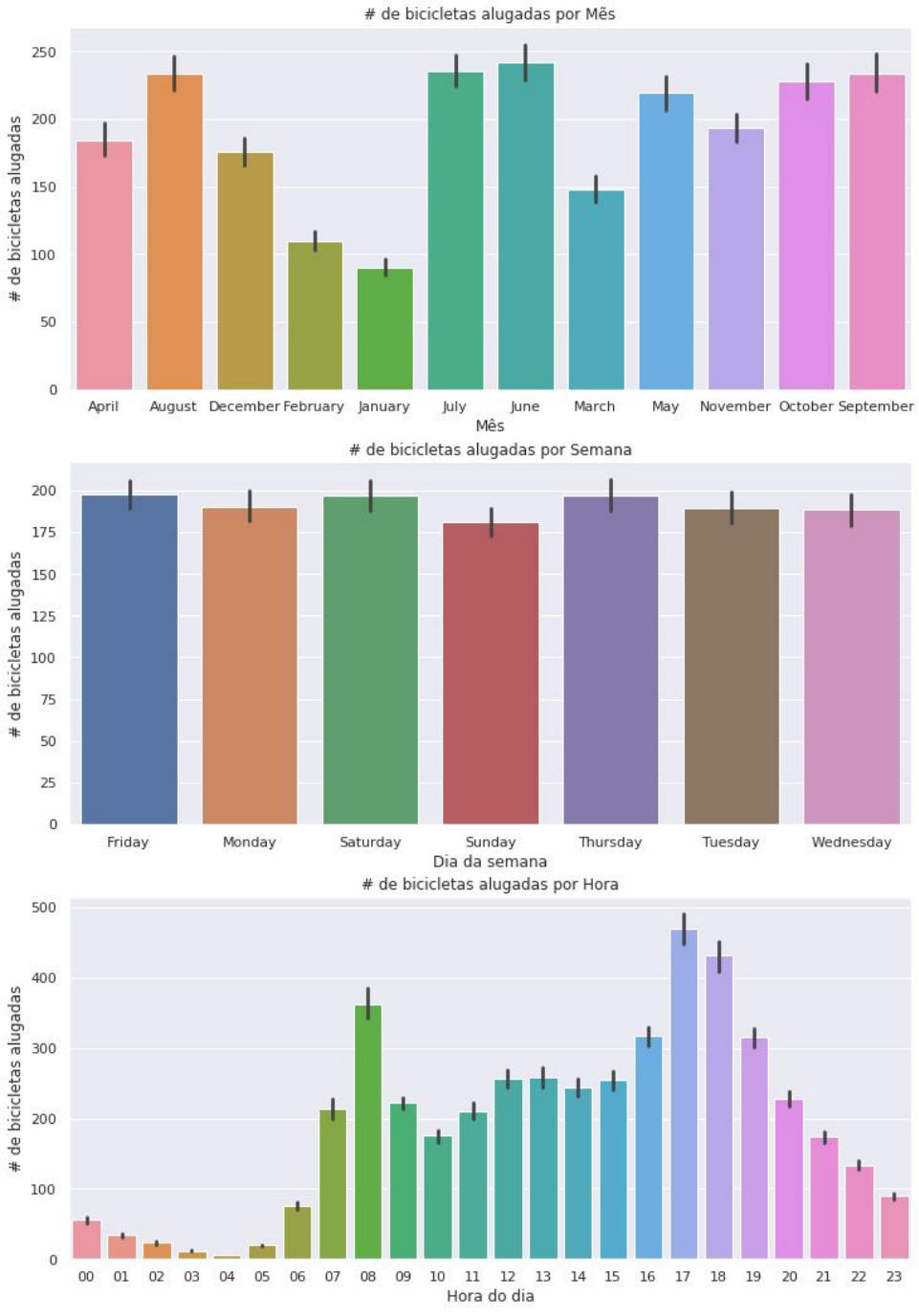
Visualização exploratória

Para todos os gráficos desta seção, foi usada a biblioteca *Seaborn* e *Matplotlib*. Vamos começar entendendo qual é a distribuição dos dados da variável resposta (count). Para isso, utilizarei a função *distplot* do *Seaborn*. Do lado direito, temos a versão dessa variável na sua escala original. Do lado esquerdo, temos a versão dessa variável na escala logarítmica. Issa última ajudar a ter uma ideia melhor do formato da distribuição que estamos trabalhando.



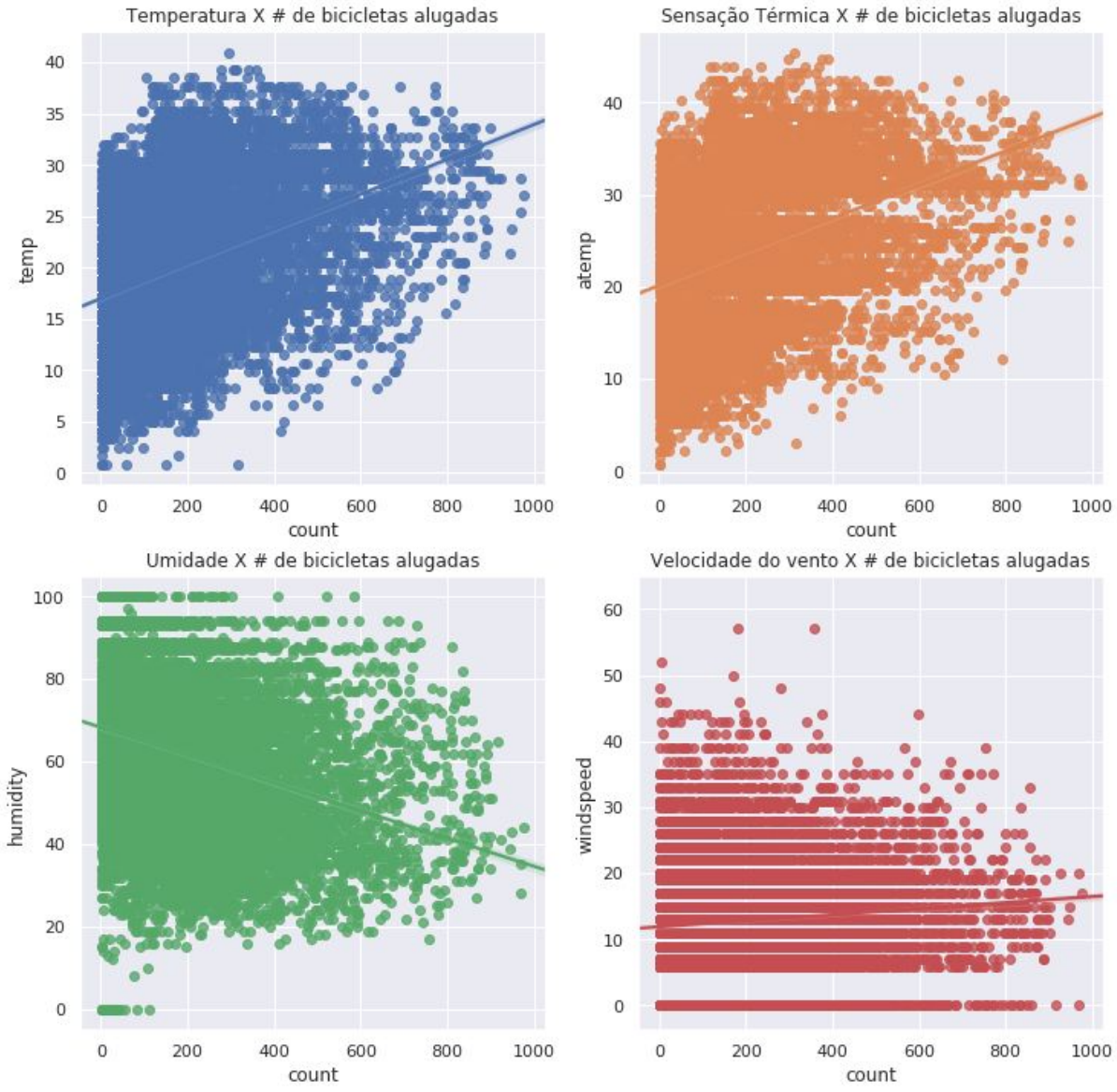
Vemos que é a variável count é decrescente. Isso significa que é mais raro existir um momento em que 800 bicicletas estão alugadas em relação a um momento em que existem 100 bicicletas alugadas. No gráfico da direita, podemos notar que na maior parte do tempo há entre ~100 e ~300 bicicletas alugadas, como podemos ver pelo pico entre 2 (10^2) e 2.5 (10^2.5).

Abaixo, vamos entender como as variáveis de tempo (Hora, Dia da Semana, Mês) impactam no aluguel de bicicletas.



Nos meses de Janeiro e Fevereiro, o número de bicicletas alugadas é consideravelmente menor do que nos meses de Junho e Julho. Provavelmente isso se deve ao fato de ser inverno no hemisfério norte nos meses de Janeiro e Fevereiro, impactando no apetite dos usuários a sair de cada para andar de bicicleta. Em relação aos dias da semana, parece não existir diferença considerável visto que há interseção dos intervalos de confiança. No terceiro gráfico, podemos notar que a demanda é mais alta no início da manhã (~8 horas) e no final da tarde (17 e 18 horas). Provavelmente isso ocorre porque essas bicicletas estão sendo usadas para deslocamento casa-trabalho. Portanto, nesses horários em que as pessoas estão indo ou saindo do trabalho, a demanda é maior.

Logo abaixo, vamos explorar as variáveis numéricas.



Parece que tanto para a temperatura quanto para a sensação térmica mais alta, a demanda de bicicletas aumenta. Podemos observar essa tendência analisando os dois

primeiros gráficos da figura acima. O terceiro gráfico nos mostra uma tendência inversa, ou seja, a demanda de bicicletas aumenta quando a umidade tende a diminuir. Já em relação a velocidade do vento. Parece que é melhor que ela esteja estável e baixa. Isso faz sentido dado que a velocidade do vento pode atrapalhar os ciclistas.

Todas as variáveis analisadas acima parecem ser boas preditoras para o número total de bicicletas alugadas. Dessa forma, todas elas serão usadas na etapa de modelagem.

Algoritmos e técnicas

Nesse projeto, serão usadas duas técnicas: Random Forest e Gradient Boosting. Ambas baseadas em árvores de decisão. Logo abaixo, vemos uma breve descrição dessas técnicas e suas vantagens para resolver esse problema

Random Forest (RF): Essa técnica treina cada árvore independentemente, usando amostragens aleatórias dos dados. Essa aleatoriedade ajuda o modelo a ser mais robusto que apenas uma árvore de decisão sozinha e isso ajuda a combater o overfitting. Além disso, a RF lida bem com variáveis categóricas, o que é interessante para o dataset desse projeto visto que existem diversas variáveis desse tipo.

Gradient Boosting (GB): Essa técnica treina uma árvore de decisão de cada vez, onde cada nova árvore gerada ajuda a corrigir os erros das árvores previamente treinadas. Esse processo iterativo a torna uma técnica bastante robusta.

Grid Search: O Grid Search constrói um modelo com cada combinação dos hiperparâmetros fornecidos e a partir disso encontra os hiperparâmetros que otimizam o modelo.

Benchmark

O principal Benchmark desse projeto é o Kernel com maior número de votos na plataforma kaggle (https://www.kaggle.com/viveksrinivasan/eda-ensemble-model-top-10-percentile). Neste notebook, o autor optou por testar cinco modelos distintos. A tabela abaixo mostra cada um desses modelos, bem como o resultado da métrica utilizada (RMSLE).

Modelo	RMSLE (Treino)	RMSLE (Teste)
Linear Regression Model	0.9779	-
Regularization Model - Ridge	0.9779	-
Regularization Model - Lasso	0.9781	-

Ensemble Model - Random Forest	0.1028	-
Ensemble Model - Gradient Boost	**0.1899**	**0.41 (Test Score do Kaggle. Este resultado está no final do kernel usado como benchmark)**

Podemos observar que o modelo testado e avaliado pelo Kernel do Kaggle foi o último **(Ensemble Model - Gradient Boosting), com RMSLE de 0.1899 em treino e 0.41 em teste.** Portanto, esse será o meu Benchmark, o qual tentarei melhorar ou chegar o mais próximo possível. O cálculo dessa métrica está descrito e justificado na seção **Métricas** deste relatório. O cálculo dessa métrica fornecido pelo autor foi feito com base no conjunto de dados de treinamento.

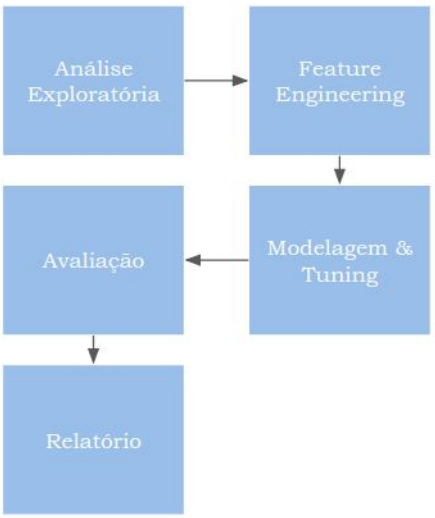
III. Metodologia

Pré-processamento de dados

Foram necessários cinco pré-processamentos: **1. Data:** A data original está no formato datetime e para facilitar a análise exploratório, bem como a modelagem, ela foi quebrada em outros atributos, como ano, mês, dia, hora. Além disso, foi usada a lib calendar para criar um coluna referente ao dia da semana. **2. Variável season:** (Estações do Ano). Ela foi transformada de valores numéricos para as respectivas categorias (Sprint, Summer...). **3. Variável weather:** Foi feita uma transformação semelhante a season. **4. One Hot Encoding:** Trata-se de uma técnica na qual as variáveis categóricas são transformadas de modo que cada categoria se torna uma nova coluna no dataset. Como a cardinalidade das variáveis categóricas deste dataset é baixa, então essa técnica pode contribuir com melhores predições. **5. Standardization:** Trata-se de uma técnica usada para re-escalar as variáveis numéricas sem alterar seu comportamento ou natureza . Por exemplo, neste projeto podemos ver que a escala da umidade é diferente da escala da temperatura. Portanto, parece ser uma técnica adequada para este tipo de dataset.

Implementação

O ciclo de vida do projeto pode ser descrito pelo diagrama abaixo:



Análise Exploratória: Nessa etapa, serão analisadas detalhadamente cada uma das variáveis fornecidas a fim de compreender melhor o problema e identificar quais variáveis serão melhores preditoras. Além disso, análise de missing values, outliers e a distribuição da variável resposta são indispensáveis nessa fase.

Feature Engineering: A partir das análises realizadas na etapa anterior, possivelmente chegarei em algumas transformações das variáveis originais que farão maior sentido para a solução do problema. Esta etapa consiste em construir essas variáveis transformadas para que sejam usadas no modelo.

Modelagem & Tuning: Nessa etapa, serão aplicados os algoritmos de aprendizado de máquina supervisionado (*RandomForestRegressor, GradientBoostingRegressor*). Além disso, serão testadas técnicas de tuning do modelo.

Avaliação: Uma vez que os modelos foram criados, essa etapa consiste em avaliar a performance de cada um dos algoritmos, bem como compará-la com a performance do modelo de benchmark proposto.

Relatório: Será feito um relatório detalhado sobre todo o processo de desenvolvimento do modelo e como ele pode ser usado para resolver o problema de negócio.

Refinamento

Para avaliação dos resultados, o dataset original (train.csv) foi dividido entre treino e teste de tal maneira que 70% dos dados foram usados para treino e 30% dos dados foram usados para teste.

O processo de aperfeiçoamento dos modelos ocorreu por meio do teste de hiperparâmetros. Os parâmetros de ajuste principais foram o número de árvores (*n_estimators*), profundidade máxima da árvore (*max_death*) e a taxa de aprendizado

(*learning_rate*). Para a solução inicial, esses parâmetros receberam os valores 100, 3 e 0.3 respectivamente para a técnica Gradient Boosting e para a Random Forest, com exceção do último parâmetro (learning_rate) que foi usado apenas no Grandient Boosting. Levando em consideração os parâmetros iniciais explorados até aqui, montamos um **Baseline** para que pudéssemos ter como referência. Os resultado desse experimento estão organizados na tabela abaixo:

Modelo	RMSLE (Treino)	RMSLE (Teste)
Random Forest	0.6915	0.6852
Gradient Boosting	0.3375	0.3715

Com o Baseline pronto, foi feito um ajuste de hiperparâmetros via *Grid Search*. Para o **Gradient Boosting**, foram usados os seguintes hiperparâmetros:

n_estimators	*[100, 300, 500]*
max_depth	*[3, 6, 32]*
learning_rate	*[0.03, 0.1, 0.3]*

Após esse teste, verificou-se que o modelo resultante possui *n_estimators* = 500, *max_depth* = 6 e *learning_rate* = 0.1. No caso do **Random Forest**, os hiperparâmetros usados foram:

n_estimators	*[100, 300, 500]*
max_depth	*[3, 6, 32]*

Após esse teste, verificou-se que o modelo resultante possui *n_estimators* = 500, *max_depth* = 32. Em seguida, os resultados finais foram coletados e organizados na próxima seção (Resultados). Devido ao alto custo computacional do Grid Search, só foi possível utilizá-lo com até três valores para cada hiperparâmetro.

IV. Resultados

Modelo de avaliação e validação

As tabelas abaixo mostram o desempenho dos modelos antes e após o ajuste de hiperparâmetros:

Antes do GridSearch (Baseline):

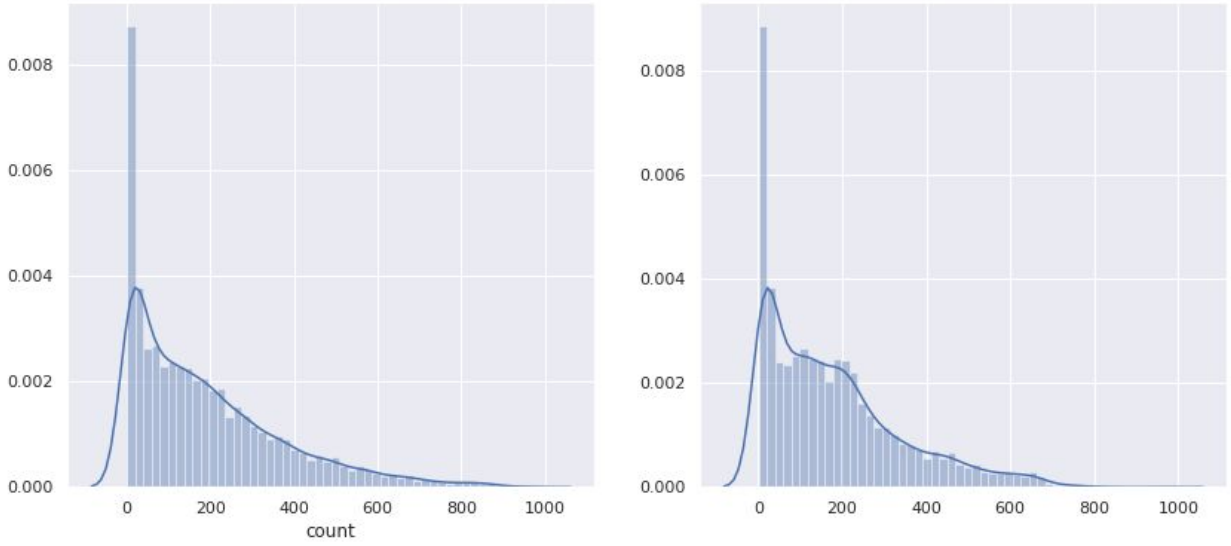
Modelo	RMSLE (Treino)	RMSLE (Teste)
Random Forest	0.6915	0.6852
Gradient Boosting	0.3375	0.3715

Após o GridSearch:

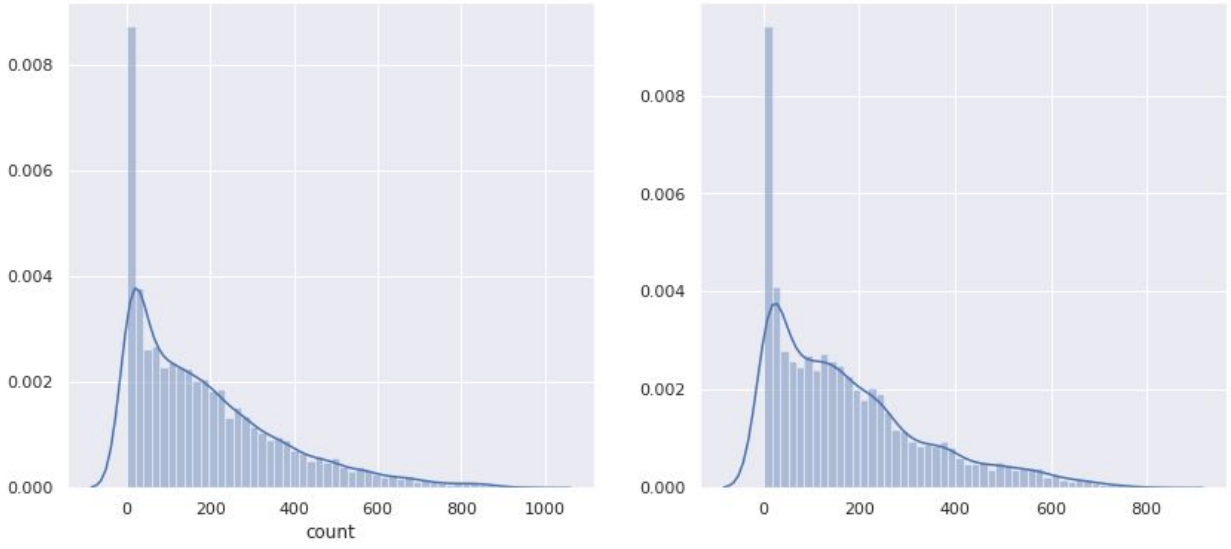
Modelo	RMSLE (Treino)	RMSLE (Teste)
Random Forest	0.1343	0.3715
Gradient Boosting	0.1488	0.3262

Vemos que a métrica RMSLE melhorou para ambos os modelos tanto para o dataset de teste quanto para o dataset de treino. Entretanto, a melhora foi mais acentuada para o Random Forest cujo RMSLE foi de 0.6852 para 0.3715 (diferença de 0.3137), em comparação com o Gradient Boosting que foi de 0.3715 para 0.3262 (diferença de 0.0453). Parece que o modelo que apresentou o menor overfitting foi o Baseline da Random Forest, visto que há a menor diferença entre o RMSLE de treino e teste. Entretanto, também foi o que apresentou a pior performance em termos do RMSLE (0.6852).

Uma possível maneira de comparar a variável resposta do conjunto de dados de treinamento com as predições realizadas no conjunto de dados de teste consiste em comparar as distribuições da variável predita (após do 20º dia de cada mês) com a variável medida (até o 20º dia de cada mês). Fizemos esse teste para ambos os modelos. O gráfico abaixo é referente ao Gradient Boosting.



As distribuições acima estão bastante parecidas. Portanto, isso significa que o modelo provavelmente está realizando predições razoavelmente proporcionais a um cenário real. Analogamente, para o modelo Random Forest nós obtivemos o resultado abaixo:



Justificativa

O modelo que apresentou menor RMSLE nos dados de teste foi o Gradient Boosting após ajuste de hiperparâmetros. Vamos comparar o resultado do melhor modelo deste projeto com o resultado do Benchmark na tabela abaixo.

Modelo	RMSLE (Treino)	RMSLE (Teste)
Gradient Boosting (Benchmark)	0.18	0.41
Gradient Boosting (Este projeto)	0.14	0.32

Dessa forma, o modelo deste projeto obteve um melhor desempenho em relação a métrica RMSLE tanto para os dados de treino quanto para os dados de teste. Apesar disso, ambos os modelos devem ter sofrido overfitting, visto que há uma diferença de RMSLE entre os dados de treino e teste. Acredito que este modelo pode ter ficado melhor que o benchmark por duas razões principais:

1. Ajuste de hiperparâmetros via GridSearch. O modelo pode ter chegado em hiperparâmetros mais adequados para esse conjunto de dados.
2. Uso de técnicas como StandardScaler, que pode ter sido relevante dado que os atributos tinham escalas bem diferentes.

Para validar se o modelo possui um boa performance para dados jamais vistos, foi utilizada a função *split_train_test* para separar uma amostra para treino e uma amostra para teste. Dessa forma, o resultado acima (0.32) mostra o RMSLE para dados jamais vistos no processo de treinamento. E o que podemos observar é que mesmo sem conhecer esses dados, o erro foi relativamente baixo.

Neste projeto, foi usado o StandardScaler (z-score) justamente para que ele fosse robusto a outliers. Dessa forma, creio que esse modelo tenha essa característica.

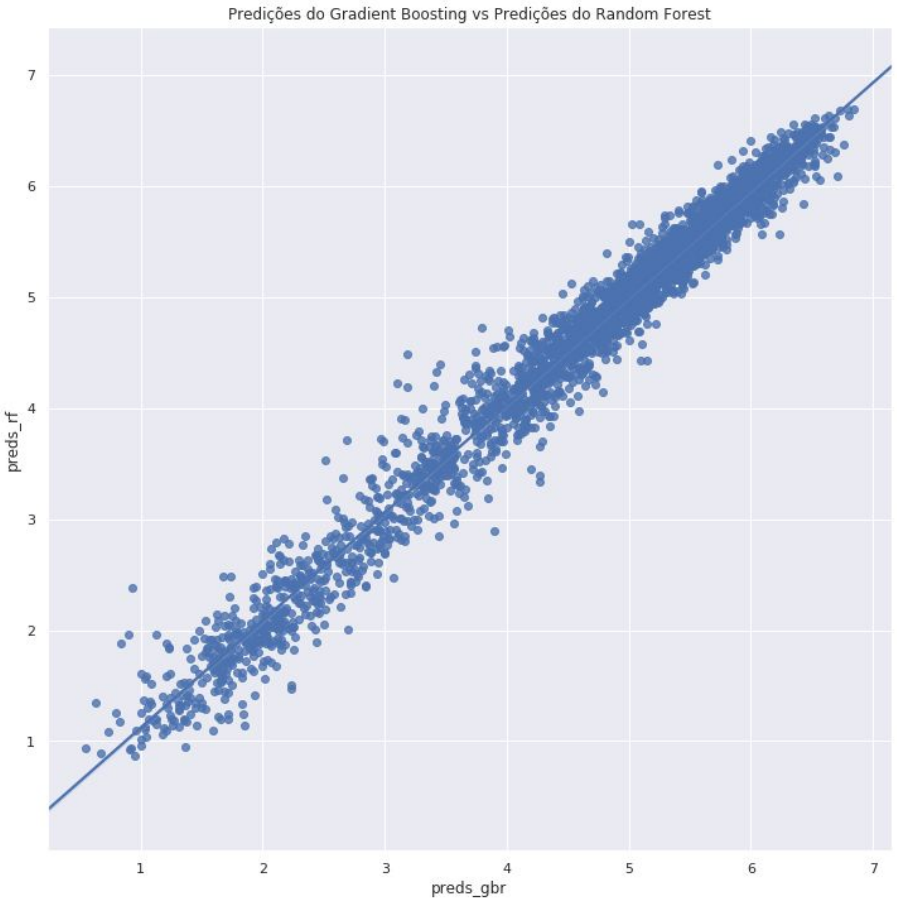
Em minha visão, podemos confiar neste modelo uma vez que observamos que o RMSLE foi baixo, assim como o overfitting.

Acho que esse modelo resolve o problema proposto de maneira suficiente. De acordo com a caráter do problema (predição da demanda de bicicletas), não há a necessidade do algoritmo ser extremamente preciso. Dessa forma, creio que esse modelo poderia ser colocado em produção. Apesar disso, caso os stakeholders do projeto queiram um modelo cujo risco é mais baixo, a melhor recomendação é o modelo Random Forest (baseline), o qual obteve a menor disparidade de predições entre os datasets de treino e teste. Cuja performance não é a melhor, porém foi o que menos apresentou overfitting.

V. Conclusão

Forma livre de visualização

Uma forma de visualização das predições de ambos os modelos testados é fazer um *scatterplot,* como o mostrado na figura abaixo. Cada eixo representa as predições de um modelo. Eixo x representa o Gradient Boosting e eixo y a Random Forest.



Vemos que ambos os modelos estão fazendo as predições de maneira mais ou menos parecida. Além disso, parece que a dispersão é um pouco maior para valores menores.

Reflexão

O problema me chamou a atenção logo que o vi quando estava buscando pelo desafio na plataforma Kaggle. Me chamou a atenção por ser um problema que pode impactar na cadeia logística de fornecimento de recursos, nesse caso as bicicletas. Nas fases de

preparação do ambiente de desenvolvimento escolha das bibliotecas a serem usadas não houve nenhuma novidade. Já em relação a fase de análise exploratória, não houve novidade no sentido de ferramental estatístico, porém acabei explorando mais as ferramentas de visualização de dados (matplotlib e seaborn). Houve bastante aprendizado em relação a essas ferramentas. Na Feature Engineering e Modelagem, também foram etapas sem muito mistério. Com exceção do GridSearch, que mostrou-se muito demorado em sua execução. Fazendo com que eu tivesse que simplificar o problema e os hiperparâmetros utilizados.

O aspecto que considerei mais difícil desse projeto foi a escrita do relatório e o tempo de treinamento. Esse material é bastante completo e mesmo após a finalização dos códigos no Jupyter Notebook, as perguntas deste relatório me fizeram voltar várias vezes para o Notebook com o intuito de entender melhor o que havia feito.

A solução final está alinhada com a resolução do problema. A métrica chave apresentou um resultado satisfatória, como explicado na seção de Resultados.

Melhorias

Dentre as principais melhorias para esse projeto, destacam-se a criação de mais modelos e teste de mais hiperparâmetros. É possível que um testes mais sofisticados com hiperparâmetros levasse a uma performance melhor. Além disso, poderiam ser exploradas outros tipos de transformações dos dados (Feature Engineering). Poderiam ser criados Pipelines de modelos usando o próprio sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html).

Por fim, existe vários outros atributos que podem impactar no aluguel de bicicletas, tais como a região geográfica, número de habitantes de uma determinada região, índice de acidentes e assaltos, preço do aluguel, entre outros. Dessa forma, poderia ser realizado um enriquecimento da base de dados para que essas informações fossem analisadas e eventualmente integradas ao modelo.