Séries Temporais e Machine Learning

MESZ, Lucas

11 de junho de 2019

Introdução

O tema escolhido para este projeto é em função da utilidade para a tese no mestrado acadêmico em Finanças pelo IAG PUC-Rio. A tese em si ainda não foi definida, contudo o trabalho envolverá estimativas de valor presente de investimentos no ramo de Óleo e Gás. Para isso, é necessário o conhecimento das séries temporais de tipos de óleo comercializados e seus derivados. Desta forma, a disciplina Ciência de Dados -INF2420- e este projeto serviram de arcabouço para treino de métodos tradicionais de estimativa em finanças, prática com programação, organização da informação e implementação de novas técnicas de aprendizado de máquina. Neste documento, só serão expostos as simulações envolvendo aprendizado de máquina. As rotinas geradas para este trabalho foram feitas em Python e estão no meu link do github.

As referências para construção deste documento foram: *i*) Materiais do DataCamp, em especial o curso Machine Learning for Finance in Python e o curso Machine Learning for Time Series in Python e *ii*) rotinas e materiais da plataforma Quantopian.

1 Dado Bruto e Análise do Dado

A base de dados é do site Quandl. Trata-se de um dado compilado de contratos futuros (CME) de óleo tipo WTI em um lag de um mês (CL1). O download dos arquivos foram feitos no dia 03/06/2019 .

Como foi utilizado um api do Quandl para baixar e esta série é atualizada dia a dia, decidiu exportar pelo *pandas* um csv e importá-lo para um *DataFrame*. O dado não precisou ajustar o *index* em *datetime*. Também não foi necessário adequar os nomes das colunas, já que eram curtos e fáceis de associar. Para séries temporais no *pandas*, ajustar o *index* e as colunas facilita o trabalho. Este csv também foi carregado no github.

Os dados brutos foram analisados em conjunto.Como pode ser visto na figura (1), são todas as colunas referentes ao óleo WTI, com os preços Open, High, Low, Last e Settle e volumes de transações como Volume e Previous Day Open Interest . Como medidas de previsão e de *features* de previsão, foram utilizadas somente atributos de Last, Volume e Previous Day Open Interest.

Date	Open	High	Low	Last	Change	Settle	Volume	Previous Day Open Interest
1983-03-30	29.01	29.56	29.01	29.40	NaN	29.40	949.0	470.0
1983-03-31	29.40	29.60	29.25	29.29	NaN	29.29	521.0	523.0
1983-04-04	29.30	29.70	29.29	29.44	NaN	29.44	156.0	583.0
1983-04-05	29.50	29.80	29.50	29.71	NaN	29.71	175.0	623.0
1983-04-06	29.90	29.92	29.65	29.90	NaN	29.90	392.0	640.0

Figura 1: Dado de Entrada: Head

A figura (2) mostra a série histórica do último preço ("Last") do WTI e seu volume de negociações. Nota-se grande variação de preços e volumes ao longo do tempo. Sobre o preços, nos anos 80 e 90, o preço era relativamente baixo comparado aos anos 2000. Este efeito é descrito essencialmente pelo crescimento chinês. As grandes oscilações em 2008 são devido a crise sistémica da quebra do *Lehman Brothers*. Após, em 2015, a queda nos preços é justificada pelo resfriamento do mercado e entrada do *shale gas* americano. Esse efeitos também podem ser descritos na série histórica de Volumes.

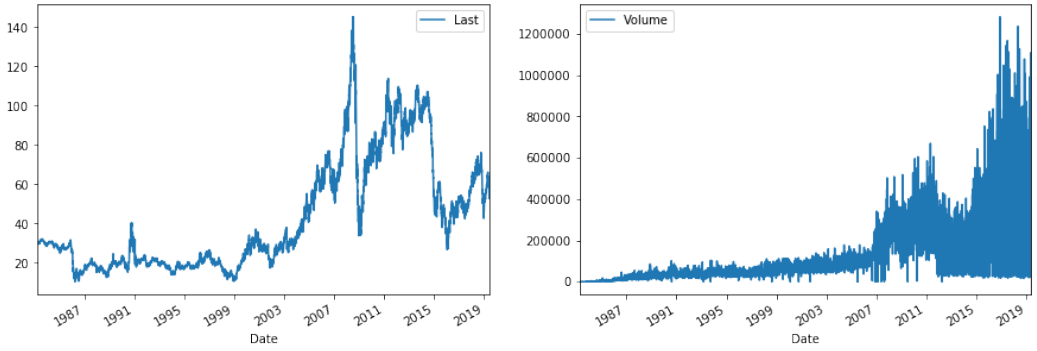

Figura 2: WTI-Série: Last e Volume

A análise do mesmo dado pode ser feita em histograma, como nota-se na figura (3). Tanto a feição dos preços ("Last") como de Volume são similares a uma função lognormal. Isso é devido a característica da subida exponencial dos preços e dos volumes de negócio ao longo da histórica.

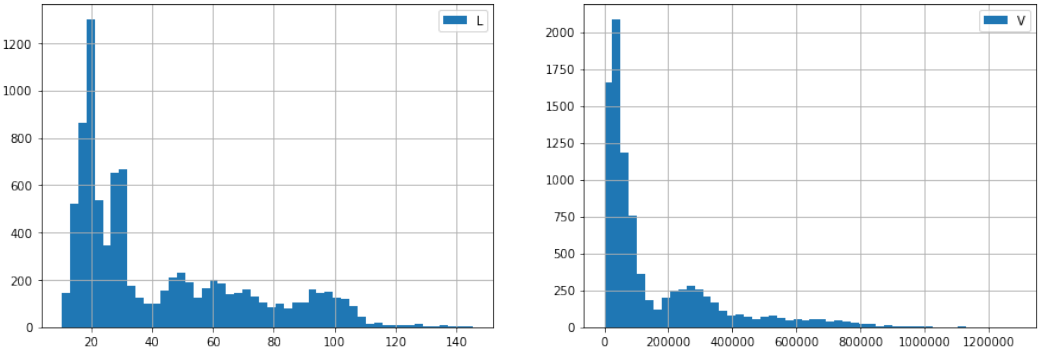

Figura 3: WTI - Histograma: Last e Volume

As informações históricas da década de 80 devem interferir pouco nos preços deste século. Inclusive, os preços médios, ao longo de toda a série mudaram de forma significativa. Por essas razão, decidiu utilizar para o aprendizado de máquina somente as informações da última década (["2009":]). As série históricas deste período podem ser vista na figura (4). Nota-se dois patamares de preços e de volumes.

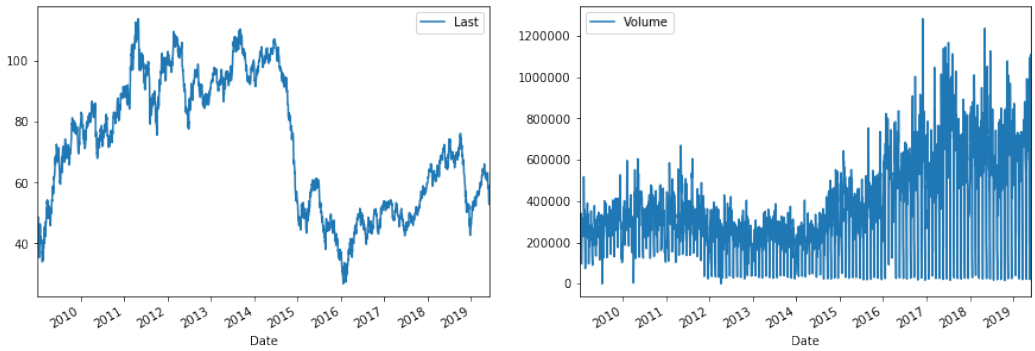

Figura 4: WTI – Série ["2009":]: Last e Volume

Esses dois patamares são notados com mais clareza nos histogramas da figura (5). Nota-se uma bimodalidade mais clara nos preços ("Last") que nos volumes.

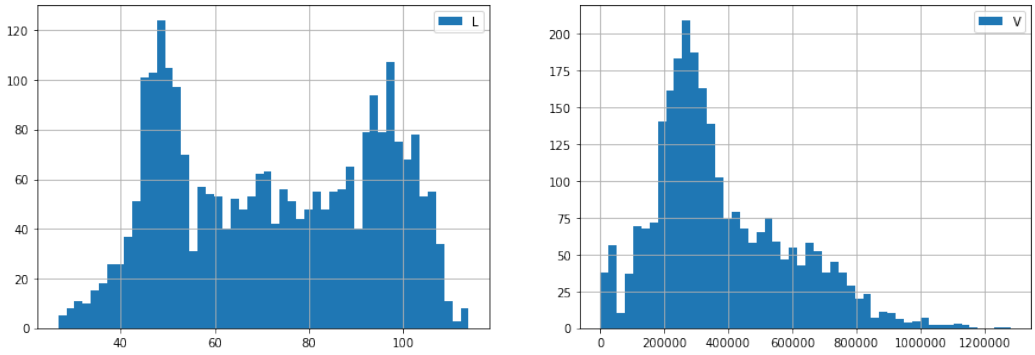

Figura 5: WTI – Histograma ["2009":]: Last e Volume

Por fim, apresento na figura (6) a descrição estatística das séries pelo atributo *describe* do *pandas*.

	Open	High	Low	Last	Change	Settle	Volume	Previous Day Open Interest
count	2624.000000	2624.000000	2624.000000	2624.000000	1329.000000	2624.000000	2.624000e+03	2624.000000
mean	72.402538	73.364733	71.352508	72.398216	0.882114	72.398765	3.761432e+05	282462.933689
std	21.735391	21.763272	21.662353	21.747723	0.765276	21.747519	2.113830e+05	152675.493254
min	27.300000	27.480000	26.050000	26.760000	0.010000	26.210000	0.000000e+00	13927.000000
25%	51.520000	52.395000	50.577500	51.580000	0.310000	51.545000	2.316555e+05	165511.500000
50%	71.875000	72.770000	70.765000	71.955000	0.680000	71.930000	3.194210e+05	288110.500000
75%	93.152500	93.932500	92.062500	93.122500	1.240000	93.112500	5.097185e+05	378235.250000
max	113.890000	114.830000	112.250000	113.930000	7.540000	113.930000	1.282869e+06	642793.000000

Figura 6: Dado de Entrada: Descrição Estatística

2 Features e Tratamentos

Foram criados treze *features* para a modelagem:

```
'd10_Last_pct','d14_SMA', 'd14_RSI','d200_RSI','d200_SMA',
'd01_Volume_pct','d14_Volume_SMA','d01_Interest_pct','d14_Interest_SMA',
'weekday_1', 'weekday_2','weekday_3', 'weekday_4'.
```

A primeira fileira possui atributos relacionados a preços. Para a geração dos modelos, tanto para as *features* como para o *target*, só utilizou o preço "Last". Foi utilizado atributo *pct-change()* do *Pandas.Dataframe* para o cálculo de variação de porcentagem. A variação de porcentagem foi de 10 dias. Para média móvel (SMA) foi utilizado o *wrapper talib* em períodos de 14 e 200 dias. Também foi gerado medida de RSI de períodos de 14 e 200. Esse é um parâmetro de força do mercado, uma razão das médias de ganhos sobre as perdas num período. Acredita-se que acima de certo valor o mercado subiu demais e deve cair e abaixo de certo número, caiu demais então deve subir.

Para os volumes, foi utilizado a variação percentual e a média móvel de 14 dias. No caso do Volume "Previous Day Open Interest"a variação de volumes foi muito brusca e ocasionou buracos nas séries. Desta forma, decidiu interpolar pelo atributo do *Pandas interpolate()*. O histograma dessa *feature* pode ser visto na figura (7) .

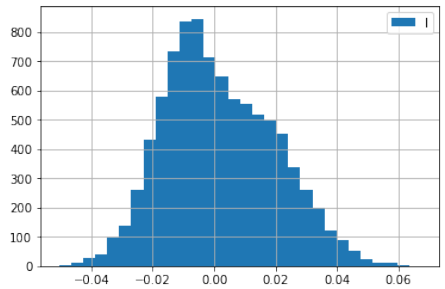

Figura 7: Histograma : Feature "d14-Interest-SMA"

Essa decisão pode ter induzido a certas respostas no aprendizado de máquina. Uma possibilidade seria tentar utilizar *dropnan()* ou $\sim$*np.isnan()*.

Como sugestão do curso Machine Learning for Finance in Python, foi criado *dummies* para os dias das semanas. Séries temporais sofrem efeitos de ciclos periódicos. Um exemplo são vendas no varejo, que se intensificam no começo do mês (recebimento do salário) e final de ano (período de festas). Para isso, recorreu ao atributo do *Pandas* chamado *get-dummies* e para evitar endogeneidade nos regressores de aprendizado de máquina.

Essas novas colunas foram concatenadas no arquivo original WTI pelo módulo *concat* do *Pandas*, para manter o mesmo index.

3 Target

No caso do objetivo, foi utilizado a variação percentual de 10 períodos na série temporal deslocada em 10 períodos. Para isso, foram utilizados os atributos *shift()* e *pct-change* do *Pandas*. Este deslocamento coloca a série temporal futura ao lado da presente. Assim, as *features* do presente tentarão prever o objetivo (*target*) de um futuro de 10 dias.

Uma forma de analisar a correlação (*df.corr()*) entre esses vetores é através do *hitmap* do *searborn*. Foi feito a correlação entres todas as features (exceto as dummies) e o target, como exposto na figura (8). Na prática, deve-se olha a primeira coluna, que possui a correlação das features com o target. Nota-se baixa correlação, por vezes negativa e bem próximas de zero.

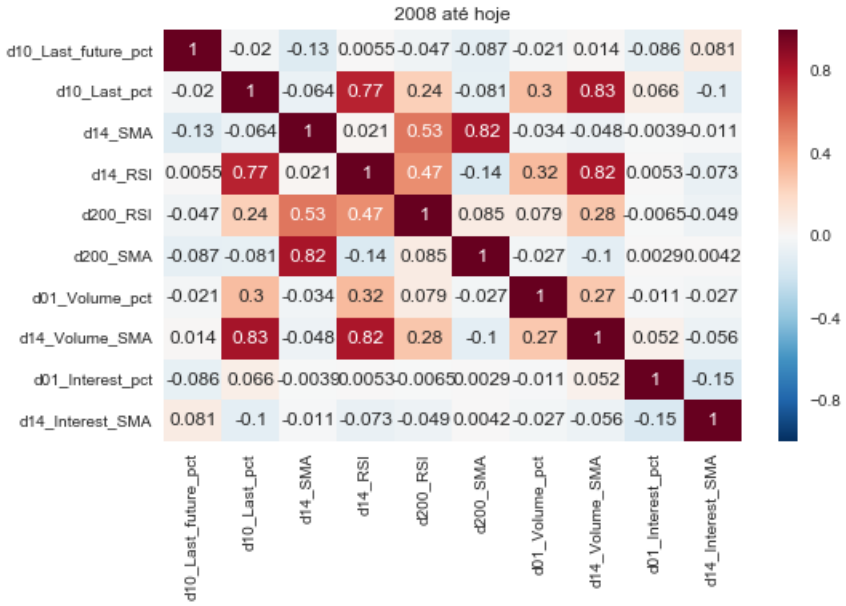

Figura 8: Hitmap : Correlação entre Target e Features

4 CrossValidation - Treino e Teste

Para incorporar esses dados nos métodos de aprendizagem de máquina, é necessário eliminar valores nulos e transformar em vetores(*array*). Assim, utilizou as funções *dropnan()* e *values()* do *pandas-Dataframe* , em todas as *Features* e *Target*.

A *crossValidation* normalmente é feita com um processo aleatório no dado. Porém, em séries temporais, isso pode ser um problema. Não se pode utilizar amostras do futuro para analisar o próprio futuro. Por essa razão, o método utilizado preza pela indexação das amostras.

No curso Machine Learning for Time Series in Python foi apresentado a função *TimeSeriesSplit* do api *sklearn.model-selection* para a *crossvalidation* . Nessa função, se insere quantas divisões o dado de entrada terá (*split=n*) e o treino (*train*) antecede a validação (*test*). Em séries temporais, essa forma é a ideal de trabalhar, contudo, são n resultados a analisar. Como não tenho tanta experiência na área, o resultado que analisei com essa ferramenta foi através de uma matriz. Achei contraproducente e não apresentarei nesse resumo, mas quem desejar, pode conferir no meu github.

Dessa forma, decidi fazer como no curso Machine Learning for Finance in Python, uma separação linear de todo o dado em 80% para treino e 20% para teste (figura 9).

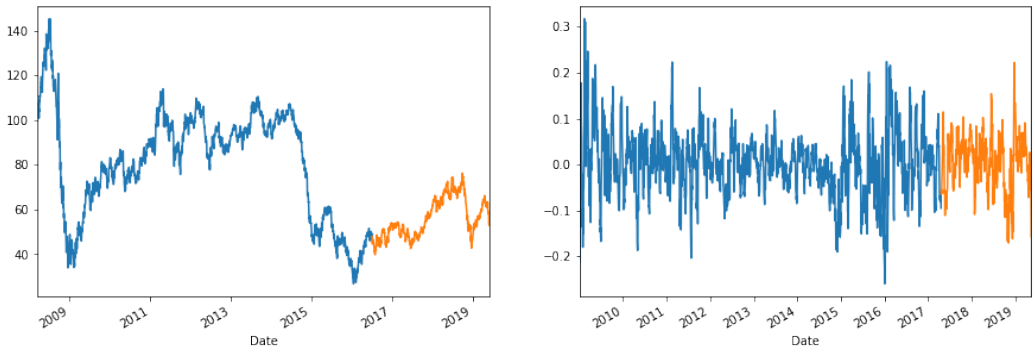

Figura 9: CrossValidation: Teste e Treino

5 RandomForest

O método RandomForest é encontrado em *sklearn.ensemble.RandomForestRegressor*. Este método combina diversas árvores de decisão e balanceia um modelo de variância (tende a ter *overfitting*) com um modelo de tendência central.

Este método tem diversos hiperparâmetros de entrada. Para isso, criou-se um dicionário para combinar os diferentes hiperparâmetros (figura 10). Como meu computador tem baixa capacidade de processamento e memória, utilizei uma variação pequena para testar a ferramenta.

```
# Create a dictionary of hyperparameters and save scores
grid={'n_estimators':np.arange(199,201, dtype=int),'max_depth':np.arange(3,6,dtype=int),
      'max_features':np.arange(10,13,dtype=int),'random_state':np.arange(40,50, dtype=int)}
test_scores=[]
train_scores=[]
```

Figura 10: Dicionário de Hiperparâmetros: RandomForest

O *RanodomForest* foi rodado segundo a rotina da figura (11). Esse fluxo basicamente descompacta o dicionário criado, gera o modelo (*fit()*) em cima do treino-*features* e treino-*target*. Depois adiciona em duas listas o R^2 do modelo aplicado ao treino (azul, da figura 9) e ao teste (laranja, da figura 9). Após o término desse *loop*, utiliza a função *argmax()* do *numpy* para achar o máximo R^2 do modelo aplicado ao teste.

```
for g in ParameterGrid(grid):
    rfr.set_params(**g)  #is "unpacking the dictionary
    rfr.fit(train_features,train_targets)
    train_scores.append(rfr.score(train_features,train_targets))
    test_scores.append(rfr.score(test_features,test_targets))



best_idx=np.argmax(test_scores)
print(train_scores[best_idx], test_scores[best_idx], ParameterGrid(grid)[best_idx])
```

Figura 11: RandomForest

Apresento diversos resultados na figura (12), sendo a primeira coluna o resultado do R^2 do treino e a segunda do teste. Todos os resultados sobre o teste são baixos. Os modelos têm baixo poder de previsão. O maior valor de teste (0.0406199) foi rodado com hiperparâmetros ruins, como n-estimador de 1 e max-depth de 1. Nota-se que o R^2 do treino desse modelo é muito baixo. Isso leva a crer que foi sorte o bom resultado do teste.

```
0.036361982295715856 0.015301877288843468 {'random_state': 50, 'n_estimators': 5, 'max_features': 12, 'max_depth': 1}
0.7342333787337771 -0.19803400659842185 {'random_state': 4, 'n_estimators': 199, 'max_features': 10, 'max_depth': 9}
0.024622223606516114 0.040619989028436554 {'random_state': 43, 'n_estimators': 1, 'max_features': 4, 'max_depth': 1}
0.04517503722444827 0.010548411226973031 {'random_state': 2, 'n_estimators': 151, 'max_features': 10, 'max_depth': 1}
0.21694942827352637 0.006158616519617577 {'random_state': 49, 'n_estimators': 199, 'max_features': 11, 'max_depth': 3}
0.20781493328160405 0.006337422682819116 {'random_state': 45, 'n_estimators': 200, 'max_features': 10, 'max_depth': 3}
```

Figura 12: Resultados: RandomForest

O resultado adotado foi o último, com hiperparâmetros:

`{'random_state': 45, 'n_estimators': 200, 'max_features': 10, 'max_depth': 3}`

Uma saída interessante do modelo é a relevância (*rfr.importance-*) dos parâmetros de entrada para a previsão (figura 13). Nota-se que há maior relevância da *feature* d14-SMA, compatível com a matriz de correlação da figura (8). É seguido de medidas de força de mercado RSI. Esses são atributos muito utilizados em análise técnica. Na conclusão deste trabalho, será apresentado um comentário sobre esses atributos e o princípio de eficiência de mercado. Por fim, vê-se que as medidas dos dias da semana são insignificantes para previsão.

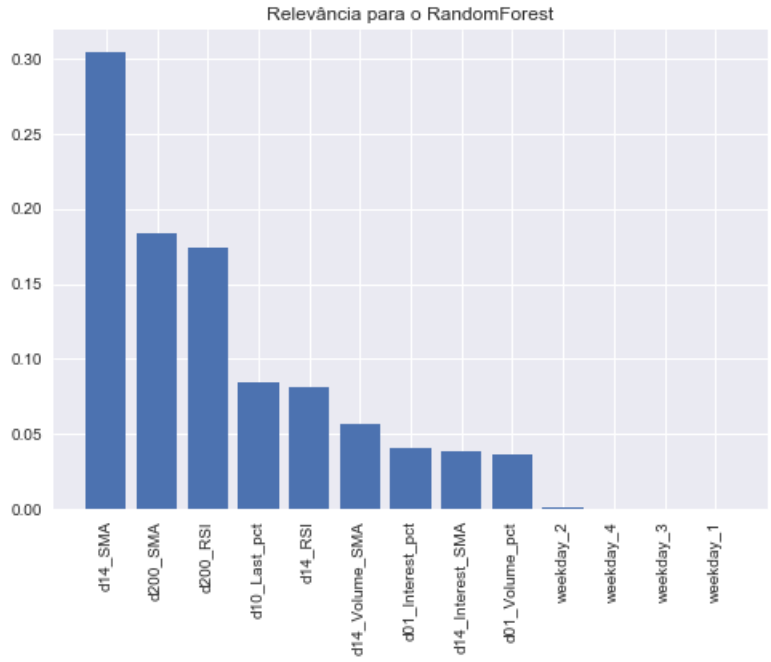

Figura 13: Gráfico de Barras: Relevância RandomForest

6 Gradient Boosting

A implementação do Gradient Boosting é bem parecida ao *RandomForest*. Esse pode ser encontrado em *sklearn.ensemble.GradientBoostingRegressor*. *Gradient Boosting* é uma classe de modelo de aprendizado que funciona como algoritmo de classificação em série, retrabalhando os erros, assim diminuindo o erro de predição.

A quantidade de hiperparâmetros é maior que o RandomForest. Foi utilizado a mesma técnica de dicionário para combinar as diferentes possibilidades (figura 14). A implementação é bem parecida com o *RandomForest*, por isso não será apresentada.

```
gbr= GradientBoostingRegressor()
grid_b={'n_estimators':np.arange(199,201, dtype=int),'max_features':np.arange(10,13,dtype=int),
        'random_state':np.arange(40,50, dtype=int), 'learning_rate':[0.01], 'subsample':np.arange(0.06,0.8,0.03)}
train_scores_b=[]
test_scores_b=[]
```

Figura 14: Dicionário de Hiperparâmetros: Gradient Boosting

Os R^2s com dados de treino e teste estão na figura (15). Nota-se melhor resultado de teste (0.01159), melhor que do *RandomForest*, ainda com o mesmo nível de R^2 de treino.

```
0.2034534547585629 0.008345592623603348 {'subsample': 0.06, 'random_state': 40, 'n_estimators': 200, 'max_features': 10, 'learn
ing_rate': 0.01}
0.2923408581372716 -0.019856049655030628 {'subsample': 0.6, 'random_state': 40, 'n_estimators': 199, 'max_features': 12, 'learn
ing_rate': 0.01}
0.25326561519992896 0.011591366178915874 {'subsample': 0.18, 'random_state': 40, 'n_estimators': 199, 'max_features': 12, 'lear
ning_rate': 0.01}
```

Figura 15: Resultados: Gradient Boosting

Os hiperparâmetros utilizados foram:

```
{'subsample': 0.18, 'random_state': 40, 'n_estimators': 199,
 'max_features': 12, 'learning_rate': 0.01}
```

O nível de relevância das variáveis também pode ser analisado (figura 16). Nota-se o resultado mais elevado e mesma configuração dos dois primeiros, com alteração das outras variáveis de volume de transação e preços. As variáveis *dummies* também se mostraram insignificantes.

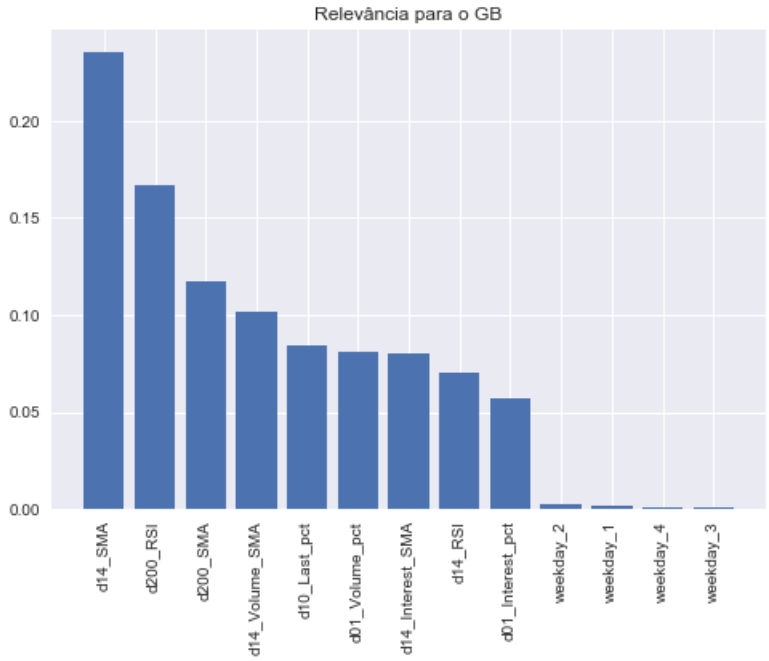

Figura 16: Gráfico de Barras: Relevância Gradient Boosting

Por fim, Apresento a comparação da predição no grupo de treino e de teste (figura 17). No treino, a predição acerta principalmente a tendência central, mas com um certo grau de oscilação. Dá para notar esse efeito de oscilação em torno da amostra 1500, acompanhando bem esta variação. Essa quebra brusca de porcentagem representa a queda dos preços no ano de 2015. Já no grupo de teste, a predição segue basicamente a tendência central, com pouquíssimo desvio.

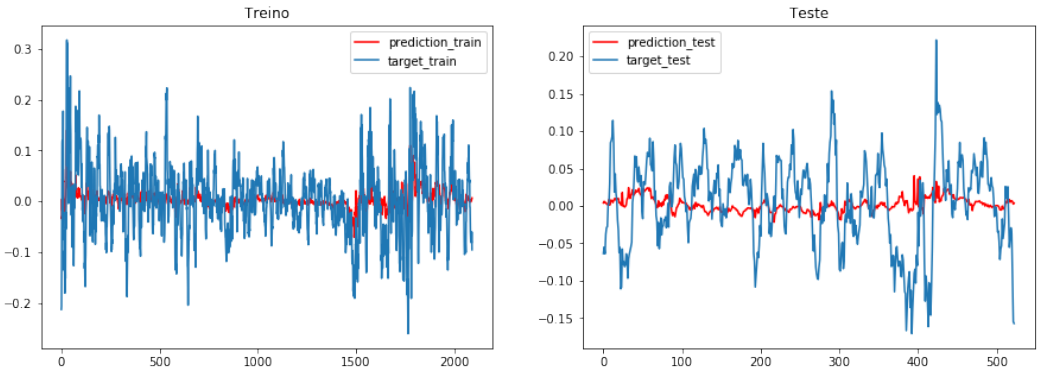

Figura 17: Teste e Treino pct-change(10) -Predição e Objetivo- Gradient Boosting

Outra forma de observar é através de um gráfico de pontos (*scatter*) (figura 18). Este gráfico coloca na abcissa a predição e na ordenada o objetivo, os treinos em laranja e em verde os testes e uma linha de 45º. Se os pontos estivessem sobre a linha preta, teríamos um modelo ideal. Veja que está muito longe disso. Nota-se que o treino está mais orientado que a nuvem de pontos do teste. Também é possível notar uma quantidade maior de *outliers* nas amostras de treino. Mas em nenhuma das nuvens a correlação é negativa e tão pouco os pontos são dispersos.

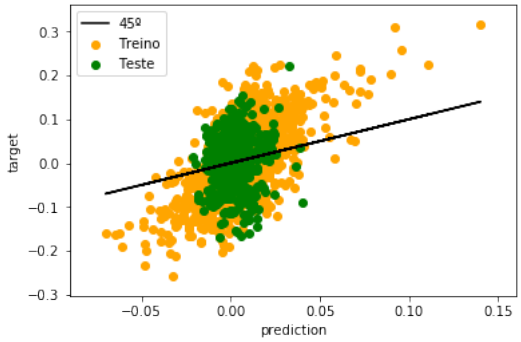

Figura 18: Scatter -Treino e Teste pct-change(10) -Predição e Objetivo- Gradient Boosting

7 Conclusão

A modelagem de séries temporais é um desafio na área de finanças. O uso de aprendizado de máquinas não é inócuo, mas por si só não trará a resposta desejada. Neste trabalho, utilizou basicamente os dados do próprio mercado. Se a hipótese de eficiência de mercado for forte, todo o preço em tempo t_0 representa todo o conhecimento naquele instante, e isso independente da informação de tempo t_{-1}. Essa visão levaria a crer que técnicas que se valem de informações do passado, como por exemplo análise técnica de gráficos, não funcionariam.

As médias móveis e RFI são muito utilizadas em análise técnica. Inclusive, justamente essas, através dos modelos *RandomForest* e *Gradient Boosting* , se mostraram mais úteis para predição de variação futura. Ainda assim, notou uma baixa aderência aos modelos, principalmente nas predições dos testes. A figura (18) sintetiza essa falha na previsão.

Ao final do curso Machine Learning for Finance in Python foi recomendado inserir outras *features* ao modelo. O mercado de óleo e gás é muito rico de informações. Poderia adicionar uma função de sentimentos com *twitters* e redes sociais, adicionar informações de balanços contábeis, projeções de análise fundamentalistas, informações da OPEP e outros. Além desse exercício, deve-se testar outros modelos e hiperparâmetros. Sem dúvida, com mais experiência nessas ferramentas, consegue-se melhores resultados com o mesmo dado.