Universidade Federal de Minas Gerais
Escola de Engenharia
Departamento de Engenharia Elétrica
Engenharia de Sistemas

Detecção de novidades e suas aplicações em NTLs

Uma solução por regressão e fluxo contínuo de dados

Igor Dias Cangussu

26 de setembro de 2020

Sumário

Resumo 4

Abstract 4

1. Introdução 5

2 - Metodologia 5

2.1 - Base de Dados 6

2.2 - ETL 7

2.2.1 - Extração de Características 11

2.3 - Machine Learning 13

2.3.1 - Árvore de Regressão 14

2.3.2 - Fluxo contínuo de dados 15

2.3.3 - Hoeffding Tree 16

2.4 - Fluxo Completo e modularização em serviços 18

3 - Resultados 20

3.1 Intervalo Padrão 21

3.2 Análise qualitativa 23

3.3 Performance 28

4 - Lições aprendidas 31

4.1 - Clustering 31

4.2 - SciKitLearn Random Forest e LightGBM 36

4.2 - Apache Spark 37

5 - Conclusão 39

6 - Referências 40

Resumo

Dados são um dos recursos mais valiosos dos dias de hoje e estão sendo gerados por diversos dispositivos a todo momento. A relevância de conseguir utilizá-los de maneira que se possa ter *insights* importantes para o negócio se torna ainda maior quando se trata de detectar perdas não técnicas na distribuição de energia elétrica que tem impacto direto no faturamento da empresa. Este trabalho tem por objetivo apresentar a aplicação do conceito de fluxo contínuos de dados utilizados para treinar iterativamente uma árvore de Hoeffding em um modelo de regressão para se poder fazer predições de consumo. A partir desta então, detectar novidades e fazer uma análise qualitativa de como esta detecção de novidade pode m auxiliar na identificação de perdas não técnicas.

Abstract

Data is one of the most valuable resources today and is being generated by different devices all the time. The relevance of being able to use them in a way that can have important insights for the business becomes even greater when it comes to detecting non-technical losses in the energy distribution grid, which has a direct impact on the company's revenue. This work aims to present the application of the concept of data stream used to iteratively train a Hoeffding Tree model in order to be able to predict the consumption. Then with the predicted value detect novelties and make a qualitative analysis of how novelty detection can assist in the identification of non-technical losses.

1. Introdução

Neste trabalho, será abordada uma solução cujo objetivo é avaliar a aplicação de técnicas de detecção de novidade para o auxílio na identificação das perdas não técnicas. Foi utilizado o modelo de regressão iterativa da árvore de Hoeffding para prever novos valores de séries temporais de consumo de clientes. Foi então avaliado a discrepância entre este valor previsto e o valor real através da definição de um intervalo padrão e enfim discutido a sua implicação na detecção de novidades e a relação destas com as perdas não técnicas.

2 - Metodologia

A metodologia será apresentada de forma a especificar cada componente criado e como eles interagem desde a chegada dos dados, passando por suas transformações até chegar ao resultado final esperado que é a detecção de pontos considerados como novidades para o sistema.

O design do projeto foi criado pensando em um produto que possa ser utilizado no mercado. Muitas das decisões tomadas levaram em consideração um caso real de uma empresa, tentando adaptar na medida do possível. A figura 1 mostra o fluxo em alto nível do sistema desenvolvido. Nas sessões seguintes, cada sub-módulo será explicado com mais detalhes.

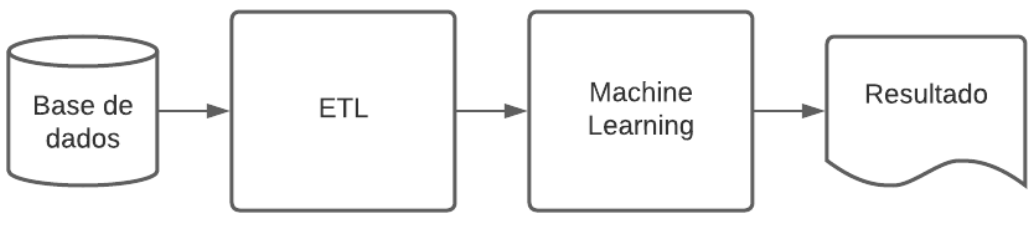

Figura 1 - Visão Geral do Sistema

2.1 - Base de Dados

A fonte de dados inicial é um CSV, ou valores separados por vírgula contendo dados de consumo de aproximadamente de 9,5 milhões de clientes na forma mostrada na figura 2. Cada linha representa um consumidor e seus valores de consumo de até 60 meses.

Figura 2 - Dados CSV

Após a etapa de ETL - Extração Transformação e Carregamento (do Inglês e*xtracting Transforming Loading)* os dados são armazenados em um banco de dados *SQLite*, escolhido por sua facilidade de uso com mínima configuração bem como compatibilidade com a ferramenta de mapeamento de objeto-relacional (*ORM* do inglês *Object-relational mapping*) *SQLAlchemy*[1]. Ela permite a abstração de entidades e manipulações de tabelas de forma facilitada, auxiliando assim na interface entre a linguagem de programação e o banco de dados. O *SQLAlchemy* também possui ótima estrutura de conexão com o banco e uma forma similar ao python de construir queries e mapeamentos de objetos. Abaixo seguem dois exemplos

[1] https://www.sqlalchemy.org/

da simplicidade e praticidade de se utilizar o SQLAlchemy inserindo e fazendo update de um registro numa tabela Users de exemplo genérica:

```
user = User(name='foo', email='foo@bar.com')
user.save()
```

```
update(users).where(users.id==5).values(name='bar')
```

2.2 - ETL

Em qualquer projeto de aprendizado de máquina, a preparação dos dados é uma etapa essencial. Em problemas reais quase nunca se terá uma base de dados bruta completamente limpa e pronta para ser utilizada no modelo. Um dos conceitos mais utilizados nesta etapa é o de ETL (do inglês *Extract, Transform and Load)*. Neste tipo de integração, é feita a extração de dados de uma ou mais fontes de dados diferentes[2]. Em seguida é feita a etapa de transformação que modela os dados no formato ideal para ser utilizado no modelo de aprendizado e finalmente o carregamento destes dados que se trata normalmente da gravação deles em uma segunda base de dados que utiliza o banco *SQLite*.

A forma da fonte principal de dados mostrada na figura 2 precisa ser inicialmente extraída e transformada. A tabela 1 mostra alguns exemplos de dados para melhor entendimento de como se deu este procedimento de transformação:

COD_INSTALACAO	OPERANDO	MES_REF	VAL_01	...	VAL_60
300000001	C_REGISTR	2018:03	204	...	270
300000002	C_REGISTR	2017:01	101	...	274
300000003	C_REGISTR	2017:07	.		376

Tabela 1

O *COD_INSTALAÇÃO* foi usado como identificador do cliente, o *MÊS_REF* é disposto de tal forma que ele é a referência do último mês da linha, sendo todos os valores precedentes relativos a *mes_ref - n* meses anteriores a ele. Finalmente temos as colunas com os valores efetivos de consumo indicadas pelos cabeçalhos *VAL_01* à *VAL_60*. O *OPERANDO* não tem relevância para o conceito desse projeto e é alvo da primeira transformação de exclusão de colunas (1).

Com o gráfico da figura 3 mostra alguns exemplos de curvas que foram escolhidas para demostrar alguns padrões diferentes de consumo. Algumas curvas possuem mudanças bruscas, outras têm poucas variações e tendência mais constante e existem até mesmo curvas somente com valores nulos. Os consumidores somente com valores nulos foram desconsiderados, entendendo que estão inativos.

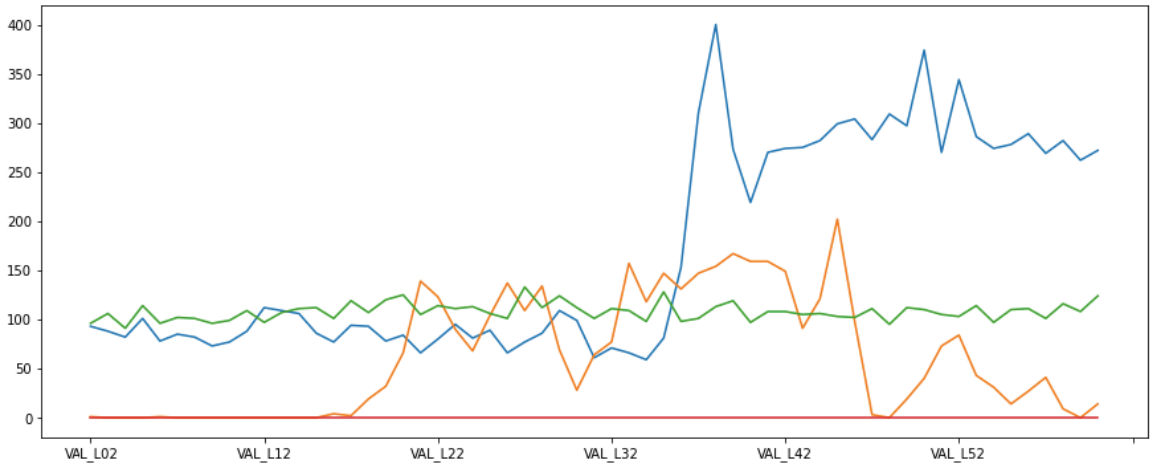

Figura 3 - exemplos de curvas diversas

O formato inicial não é interessante para ser aplicado ao modelo de previsão. O formato padrão mais usado neste caso é chamado de formato derretido (*melted*) ou explodido (*exploded*) que será definido aqui como:

Dado uma formato orientado por linhas, derreter os dados é coloca-los em formato orientado por colunas (2). Após esta transformação os dados assumem o formato mostrado na tabela 2.

COD_INSTALACAO	ANO_MES	VALOR
300000001	2013-01	93
300000001	2013-02	88
300000001	2013-03	82
300000001	2013-04	101

Tabela 2 - Exemplo de formatação por colunas

Ou seja, o que antes era um consumidor por linha com todos seus valores em diversas colunas passa a ser um mês com seu respectivo valor de consumo por linha. Atenção ao fato de que essa transformação causa um aumento significativo no espaço consumido pelos dados, contribuindo para o problema de Big Data tratado com mais detalhes nas sessões 2.3.1 e 2.3.2.

Além do formato, propriamente dito, é necessário tratar os dados faltantes (3), muito comuns em base de dados reais. Estes dados normalmente não são compatíveis com os modelos, eles são muitas vezes vazios, NaNs[2] ou outro indicador qualquer. Neste caso os dados faltantes estão representados por uma *String* ponto '.' no lugar do valor de consumo o que inclusive quebraria algoritmos que esperam receber um número para proceder com os cálculos.

[2] https://scikit-learn.org/stable/modules/impute.html#impute - SimpleImputer

Para fazer o tratamento destes dados, foi utilizado o *SimpleImputer*[3] do pacote *ScikitLearn*. Ele provê estratégias básicas para imputar estes valores nos datasets como repetir o último valor, inserir zeros ou utilizar dados estatísticos como média e mediana. Como a constância é importante no problema, e a depender da curva um zero poderia ser claramente uma novidade, a opção de substituir os valores os zeros não seria uma boa escolha, logo, para se obter algo talvez mais próximo da realidade, a média foi escolhida como forma de imputação de dados.

Recapitulando, nesta primeira fase de transformação, são feitas as seguintes transformações nos dados:

(1) Eliminação de colunas

(2) Derretimento do dados

(3) Imputação de dados faltantes

O resultado do novo esquema de dados é mostrado na figura 4.

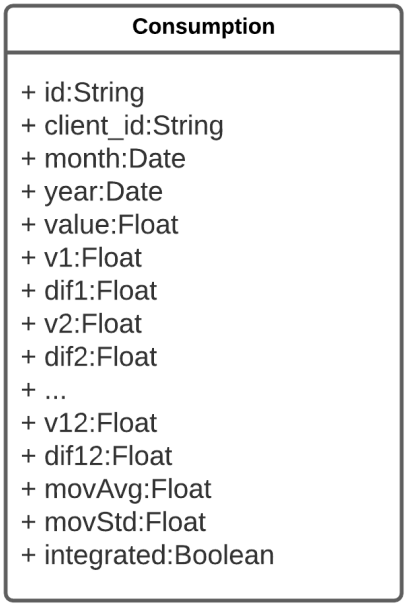

Figura 4 - Tabela Consumption

[3] https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

A segunda fase será a de extração de características.

2.2.1 - Extração de Características

Partindo do formato transformado discutido na sessão anterior, tem-se um ponto definido por um valor, uma data e um identificador do cliente. O algoritmo poderia de alguma forma tentar prever novos valores somente a partir destes dados, mas isso resultaria numa previsão muito pobre baseada somente em valores temporais e no próprio identificador do cliente. Entendendo a variedade de exemplos diferentes que a base possui e que não existem muitas repetições temporais como acontece, por exemplo, em casos de medição de temperatura, se faz necessário extrair algumas características destes dados.

Tratando de uma série temporal, é razoável assumir que os dados tem uma certa dependência dos valores anteriores. Pela suposição mais básica os valores de consumo não mudariam e simplesmente se repetiriam ao longo do tempo. Como já visto, isso não acontece, mas sim existe alguma variação ao longo do tempo. Uma **variação abrupta e inesperada no tempo** é justamente o objetivo deste trabalho: a detecção de uma novidade. Se a relação de passado e futuro da série é clara e existe uma certa variação ao longo do tempo, é razoável utilizar uma janela móvel anterior ao ponto em questão para medir a média e o desvio padrão como características desta variação bem como a derivada dos pontos anteriores [3]. A derivada é calculada simplesmente fazendo a diferença do ponto atual com o ponto anterior na forma:

$$\Delta(n) = \frac{\delta V}{\delta t} = V_{(n)} - V_{(n-1)}$$

onde:

$\Delta(n)$ = Derivada do n-éssimo termo

$V_{(n)}$ = Valor do n-éssimo termo

Foi decidido utilizar uma janela temporal de 12 meses tentando assim capturar alguma questão de sazonalidade que os dados possa ter nos dados. A versão final da modelagem resultante é mostrado na Figura 5.

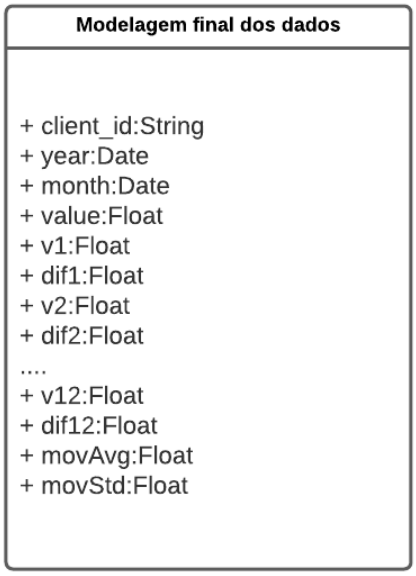

Figura 5 - Modelagem Final dos Dados

A Figura 6 ilustra os testes realizados para mostrar a relevância de cada variável para a predição utilizando o método LightGBM[4] do pacote com mesmo nome, mostraram que os valores mais próximos tem mais influência na predição do próximo valor, porém, a variável dos valores mais afastados se mostraram também importantes neste sentido. Uma vez que a solução proposta elimina o problema de espaço limitado na memória, foi escolhido

[4] https://lightgbm.readthedocs.io/en/latest/ - LightGBM

trabalhar com esta janela de 12 meses acreditando poder contribuir com a acurácia da solução.

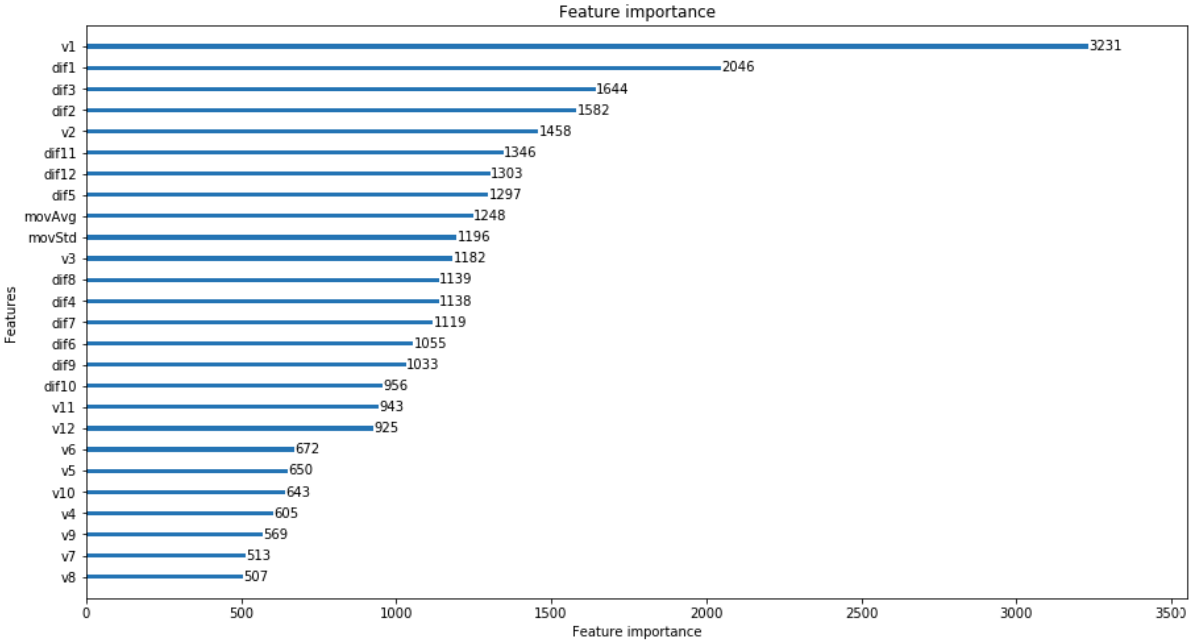

Figura 6 - importância das características

Finalizando as etapas de ETL, os dados transformados são salvos em um banco de dados para serem utilizados posteriormente.

2.3 - Machine Learning

O módulo de *machine learning* é onde os dados, já transformados, servem de entrada para o modelo de aprendizagem.

Conceitualmente, o primeiro passo para a deteção de novidade é compreender o que é considerado como "normal", para isso, deve ser utilizado alguma forma de detecção de padrão. Para este propósito, foi escolhido os métodos de predição de Time Series, sendo que o modelo aprenderá o padrão e a previsão atuaria como a representação ou

continuidade deste padrão. Caso o valor real se destoe muito do previsto, ele é considerado uma novidade.[4] Para isso foi utilizado um dos diversos métodos de predição baseados em Árvores de Regressão.

2.3.1 - *Árvore de Regressão*

O principal objetivo da árvore de regressão é prever um valor dado as variáveis de entrada. Geralmente elas são classificadas em dois modos de predição: Classificação e Regressão. Na classificação a previsão está focada em prever o pertencimento do ponto a uma classe, ja a regressão tem o objetivo de prever valores contínuos. [5] O problema aqui tratado é um problema de regressão e tem como valor final da previsão de um valor de consumo.

Na sua base, estas árvores são um aninhamento de várias regras do tipo if-else[5] de forma que ao fim, ou folhas finais da árvore, tenha-se uma tomada

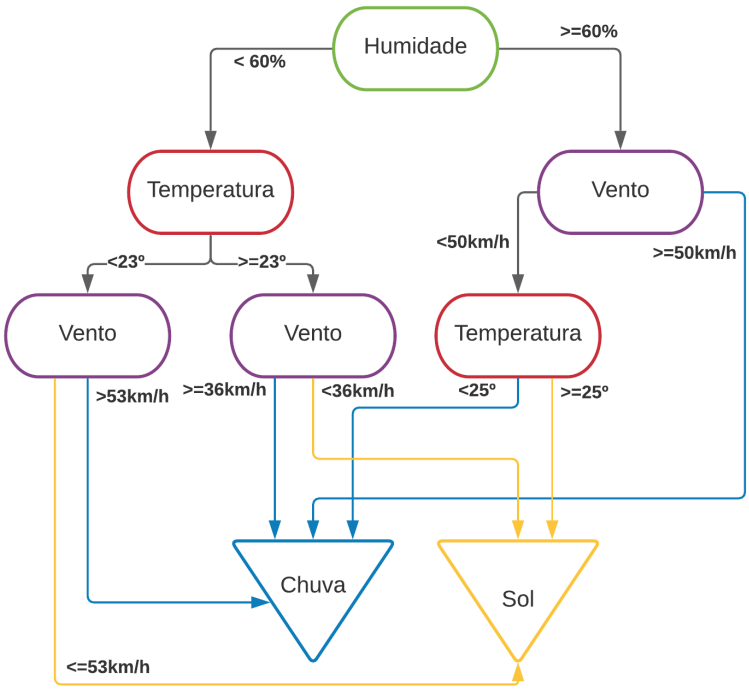

Figura 7 - Árvore de Decisão

de decisão. [6] Na figura 7 é demonstrado um exemplo de uma árvore de classificação para ilustrar o conceito de uma árvore que “prevê" se terá sol ou chuva baseado nos dados de humidade, temperatura e vento.

As árvores, porém, inicialmente necessitam ter todos os dados para a criação dos seus ramos e folhas. Neste problema, mesmo considerando a fonte de dados iniciais de tamanho limitado, as transformações escalam o tamanho da base para muito além do espaço normalmente encontrado em computadores pessoais e não seria possível carregar na memória RAM, isso torna um problema de *Big Data*, cujo conceito ainda é um pouco nebuloso, mas pode ser caracterizado, por um volume suficientemente grande de dados a ponto de não ser comportado por uma máquina somente. Para entender a solução para este problema de *Big Data*, deve-se entender o conceito de *Data Stream* ou Fluxo contínuo de dados.

2.3.2 - Fluxo contínuo de dados

O modelo de aprendizado por fluxo de dados (*Data Stream, Incremental Learning ou Online Learning*) tem ganhado muita atenção nos últimos tempos. [7] [8] Um dos motivos é a crescente geração de diferentes tipos de dados como em cliques em websites, sensores diversos e no monitoramento de tráfego. [9] Estes dados podem ser considerados como streaming pois chegam constantemente em algum intervalo de tempo formando assim um fluxo.

Este modelo tem relação com a ideia dos algoritmos que tem um aprendizado incremental com um fluxo contínuo de dados e tem como característica o fato de não ocorrer mais de uma passada nos dados durante o aprendizado. Alguns trabalhos foram feitos sobre esse assunto

considerando que também é uma boa abordagem em problemas de *Big Data*. [7] O modo de trabalho com fluxo contínuo de dados foi escolhido por possibilitar o carregamento incremental dos dados na memória e consecutivo treinamento também incremental do modelo.

Uma vez escolhida uma linha de raciocínio da solução passando pelo conceito da extração, transformação, carregamento, arvores de regressão e de data stream chega-se ao fim desta solução: a árvore de Hoeffding.

2.3.3 - Hoeffding Tree

A Árvore de Hoeffding é dita como uma árvore de decisão pronta pra previsão a qualquer momento que é capaz de aprender a partir de dados massivos de streaming. [9] Este método assume que a distribuição dos exemplos para treinamento não muda com o tempo. Esta premissa é válida para o modelo deste trabalho uma vez que se quer detectar um padrão e prever com base em uma base de dados muito grande. Mesmo existindo a possibilidade dos padrões de consumo mudarem ao longo do tempo, considerando uma base suficientemente grande com diversos perfis diferentes de cliente, o modelo cobrirá uma gama suficiente de perfis para que seja considerado um estado estacionário no tempo, ou seja, o treinamento de toda a base em um único modelo, permitirá que clientes com alto e baixo consumo sejam avaliados pelo mesmo modelo. Adicionalmente pode-se ser usado o modelo da Árvore de Hoeffding Adaptativa[6] caso o perfil dos dados desejados tenha essa característica intrínseca de mudança na distribuição. Os testes realizados com ambas as técnicas não mostraram diferenças nas predições, porém a Adaptativa tem um gasto computacional maior e por isso escolheu-se a versão tradicional.

[6] HoeffdingAdaptiveTreeRegressor - scikit-multiflow.readthedocs.io

A Árvore de Hoeffding utiliza a fronteira de Hoeffding para a construção e análise da árvore de decisão. [10] Esta fronteira no algoritmo é definida[7]

$$\text{por: } \epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}$$

onde:

ϵ: Fronteira de Hoeffding

R: O range da variável aleatória. Para a probabilidade o range é 1, para o ganho de informação o range é log(c), onde c é o número de classes.

δ: Confiança. 1 menos a probabilidade desejada de escolher o atributo correto em qualquer nó dado.

Abaixo segue um pseudo código da Árvore de Hoeffding demonstrando o uso da fronteira descrita acima no algoritmo

```
''' Input:
        Um stream de exemplos rotulados
        Um parâmetro de confiança δ
    Ɛ = Fronteira de hoeffding'''
HT = uma arvore com uma única folha (raiz)
inicializa contadores n(i,j,k) na raiz
for x,y in stream:
    HTGrow((x,y),HT,δ)
```

```
def HTGrow((x,y),HT,δ):
     ordena(x,y) para a folha l usando HT
     atualiza(contadores n(i,j,k)) na folha l
     if(os exemplos vistos até agora não são da mesma
                                          classe):
     for atributo in x:
          computa(G)
     if(G[melhor atributo] - G[segundo melhor]) > Ɛ:
```

[7] https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.trees.HoeffdingTreeRegressor.html#skmultiflow.trees.HoeffdingTreeRegressor

```
                divide a folha no melhor atributo
                for(b in ramos):
                     cria nova folha
                     inicializa contadores
```

A Árvore de Hoeffding é naturalmente incremental o que foi uma das principais vantagens encontradas para a utilização desse algoritmo nesse projeto. Além de não necessitar de carregar todos os dados na memória, ela possibilita o treinamento e aprendizado contínuo do algoritmo ao longo do tempo com a chegada de dados novos não se limitando a um novo treinamento completo economizando tempo computacional e se aproximando mais de uma solução real no mundo corporativo. Além disso, ela é assintoticamente quase idêntico ao equivalente deste método não incremental se o número de amostras for grande o suficiente. [11]

Para *tuning* dos parâmetros do algoritmo foi utilizado o *GridSearchCV* do *ScikitLearn*. Com ele pode se determinar uma *grid* de parâmetros e seus respectivos valores e ele faz uma busca exaustiva testando todas as combinações destes parâmetros e retornando os com melhor performance. Ele retornou que utilizando um *grace_period* = 1000 e um *nb_threshold* = 6 juntamente com os demais parâmetros default teria-se um melhor resultado. O *grace_period* se refere ao número de instâncias que uma folha da árvore deve observar antes de se dividir, o *nb_theshold* é o número de instâncias que deve-se observar antes de permitir o uso do *Naive Bayes*. (mais detalhes vide documentação[8])

2.4 - Fluxo Completo e modularização em serviços

[8] https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.trees.HoeffdingTreeRegressor.html#skmultiflow.trees.HoeffdingTreeRegressor

A figura 8 mostra o fluxo mais detalhado descrito nas sessões anteriores. Nela, percebe-se três fluxos de dados em vermelho, azul e verde, respectivamente: o fluxo inicial do sistema, o fluxo de novos dados em sua etapa de ETL e finalmente o fluxo de aprendizado destes novos dados.

A modularização em serviços segue a linha de tendência do mercado de desenvolvimento SOA (*service-oriented architectures*) ou arquitetura orientada a serviços, mas especificamente a de Microserviços[9]. Apesar de não ter sido implementada toda a arquitetura baseada em *microservices,* a implementação dos destes dois ilustrados na figura 8 deixa a solução mais próxima da idealmente pensada pra processos de Big Data em sistemas naturalmente distribuídos.

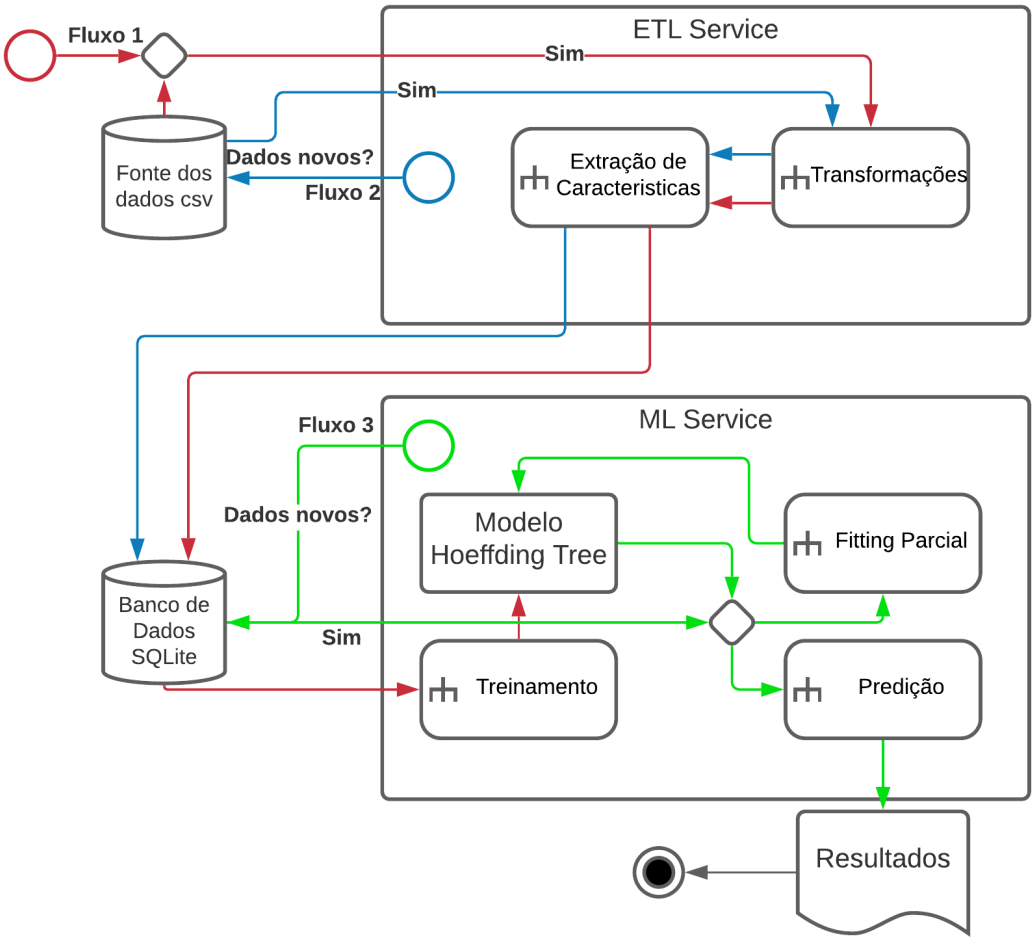

Figura 8 - Fluxo final

[9] https://www.fullstackpython.com/microservices.html

O ETL Service inicializa o processo carregando, transformando e levando os dados ao banco. O ML Service assume e treina o modelo inicial usando os dados de 10 mil curvas. O ETL então continua a inserir novos dados prontos no Banco, enquanto isso, o ML faz uma requisição ao banco verificando se existem novos dados para serem processados, preditos e incrementados no aprendizado do modelo.

Para colher os dados dos resultados, foram treinados iterativamente os dados de 5 milhões de curvas em um único modelo e para efeito de comparações, o modelo treinado foi salvo a cada 50 mil curvas. Ao final foram feito testes descritos na sessão 3 com 500 mil curvas. Neste trabalho, a fonte de dados CSV é fixa, mas, no mundo real onde os dados chegariam mensalmente, a etapa te ETL se faria responsável por verificar a base em busca de novos dados e processá-los da forma adequada para ser trabalhado pelo ML Service.

3 - Resultados

Nesta sessão é discutida a metodologia pra mensurar os resultados. Esta análise é importante de ser feita tendo em mente que se deseja avaliar como o método de detecção de novidade proposto nas sessões anteriores poderia auxiliar na tomada de decisão de detecção de fraudes ou de comportamentos anômalos de consumo dos clientes.

Essa aferição para clientes residenciais pode resultar numa detecção de fraude ou algum mal funcionamento dos componentes. Já num cliente de grande porte, além dessa abordagem, também é possível avaliar alguns picos de consumo que podem ser levados em consideração nos termos do

contrato onde muitas vezes existem multas caso as industrias ultrapassem os limites preestabelecidos por estes.

3.1 Intervalo Padrão

Normalmente chamado de intervalo de confiança da predição, aqui será chamado de intervalo padrão. Padrão porque o objetivo da predição em si, não é acertar com alta acurácia o valor real, mas sim a construção de um padrão dos dados aprendido pelo algoritmo. Não se quer medir a confiança da medição, o esperado é o ponto final da construção deste padrão onde se pode classificar um ponto como normal ou como anomalia.

Agayo (2017) [12] aplica e define o seguinte limite superior e inferior: Dado um nível de confiança α, procura-se um intervalo que contenha a porcentagem $100(1-\alpha)$ de valores para quantizar os erros dos dados. Os limites logo serão definidos como:

- **Limite inferior (τ^-):** É o percentil $\frac{100\alpha}{2}$ da distribuição da quantização dos erros associados com os vetores dos dados de treinamento.

- **Limite superior (τ^+)**: É o percentil $100\left(1-\frac{\alpha}{2}\right)$ da distribuição da quantização dos erros associados com os vetores dos dados de treinamento.

Foi utilizado um α de 0.02 nos testes.

Uma vez que o vetor de erro tem valores positivos e negativos para erros acima do valor real e abaixo respectivamente, o percentil dos limites resultam num valor negativo e num valor positivo. Somando esses valores ao valor previsto, obtemos o intervalo em questão:

$$(\upsilon + \tau^-, \upsilon + \tau^+)$$

onde υ = valor previsto

Uma vez determinado os valores $[\tau^-, \tau^+]$ uma anomalia é definida como: Se o erro entre o valor previsto e o valor real estiver dentro do intervalo o ponto será classificado como normal. Caso contrário, será classificado como anomalia.

Ou em pseudo código:

```
if error > tau_m and error < tau_plus:
    return ‘anomaly'
else:
    return ‘normal'
```

Uma mudança foi feita a partir do original proposto por Agayo (2017) [12]. Os valores de erro para calcular o intervalo foram retirados de um conjunto de teste de cerca de 1 milhão de curvas e não mais do erro do conjunto de treinamento. Além disso, tratando-se de curvas com padrões de consumo tão diferentes, foi feito um pequeno agrupamento simples a partir das médias de cada curva com o objetivo de termos intervalos diferentes com erros diferentes para cada nível de média de consumo. Esse agrupamento foi feito para se ter uma medida de erro diferente para diferentes consumos uma vez que testes feitos para todo o conjunto demonstrou erros pra mais e para menos até 6 vezes maiores do que a média da curva devido ao erro proporcional dos clientes com maior média de consumo. Em outras palavras, os consumidores com alta média tem maior tendência a ter um desvio padrão alto e consecutivo erro de previsão alto.

A figura X mostra a divisão feita levando em consideração a distribuição demonstrada na sessão 4.1.

Estes intervalos, apesar de parecerem grandes, se representam muito bem a detecção de novidade pois detecta quando um valor se destoa muito do valor previsto. Ele utiliza como base o erro no conjunto de testes de 1 milhão de pontos.

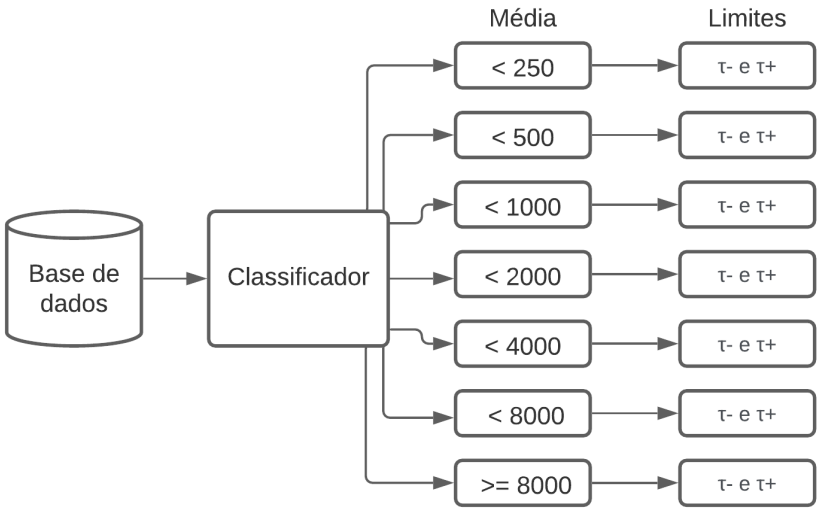

Figura X - Agrupamento pela média

3.2 Análise qualitativa

Como não se pode obter uma avaliação dos resultados de forma quantitativa uma vez que não se tem a classificação real dos pontos como normal ou não, será discutido algumas questões qualitativas a respeito dos resultados obtidos avaliando algumas milhares de curvas, seus intervalos padrão e as detecções de novidade em si. Será visto que para uma boa maioria dos casos, uma mudança abrupta é detectada. Também sera visto que para algumas curvas específicas que possuam uma variação muito grande o número de deteção é muito alto, possivelmente não refletindo a realidade. Os gráficos mostrados abaixo valores reais em azul contínuo, previsto em azul tracejado, o intervalo padrão em azul claro ao fundo e em vermelho a detecção de novidade.

A figura B mostra uma curva típica dentro do intervalo padrão previsto.

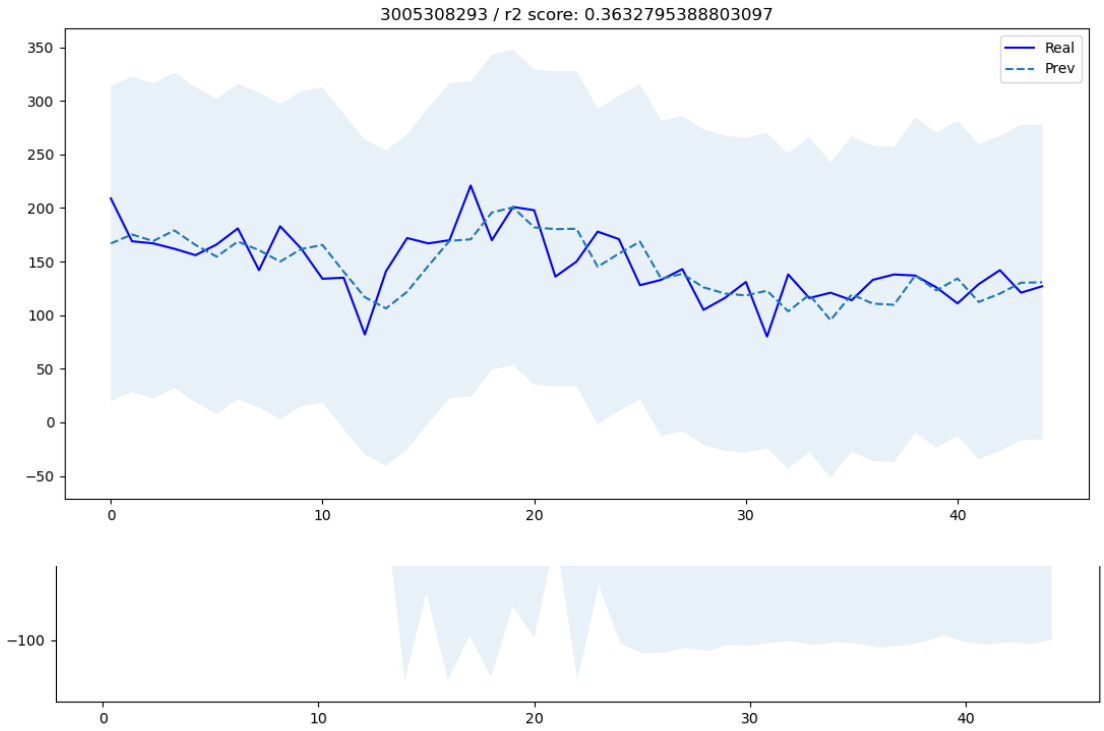

Figura X - Detecção de queda abrupta

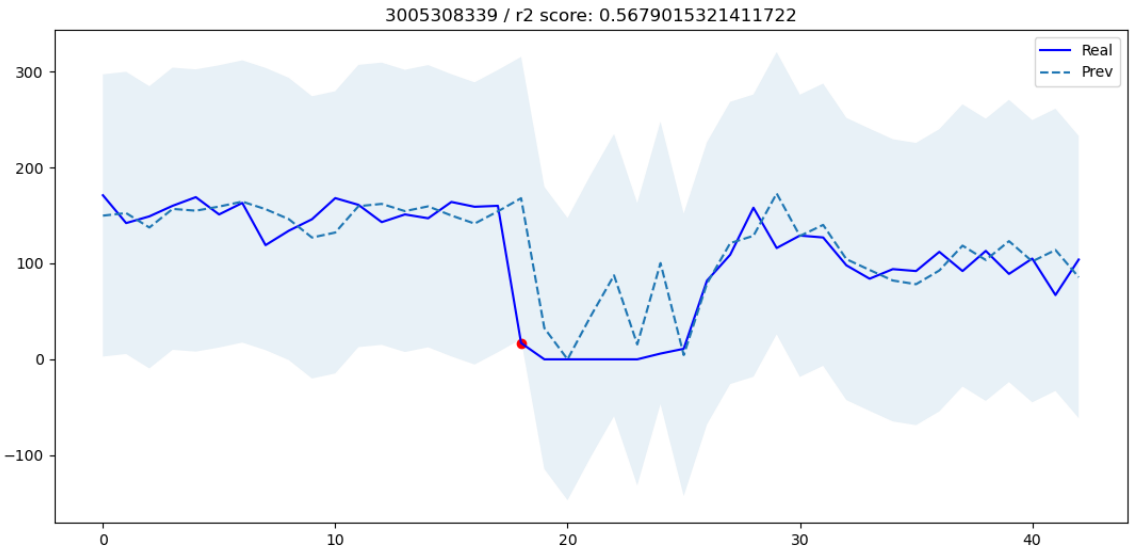

Figura X - Detecção de vale

Nas figuras X e X pode ser observado a detecção em casos de decaimentos fortes. Importante ressaltar que as detecções normalmente não são contínuas, no sentido que o modelo se adapta ao receber um valor baixo e a previsão passa a seguir este padrão. Em outras palavras a detecção acontece na mudança e não ao longo de todo o percurso de baixa ou queda.

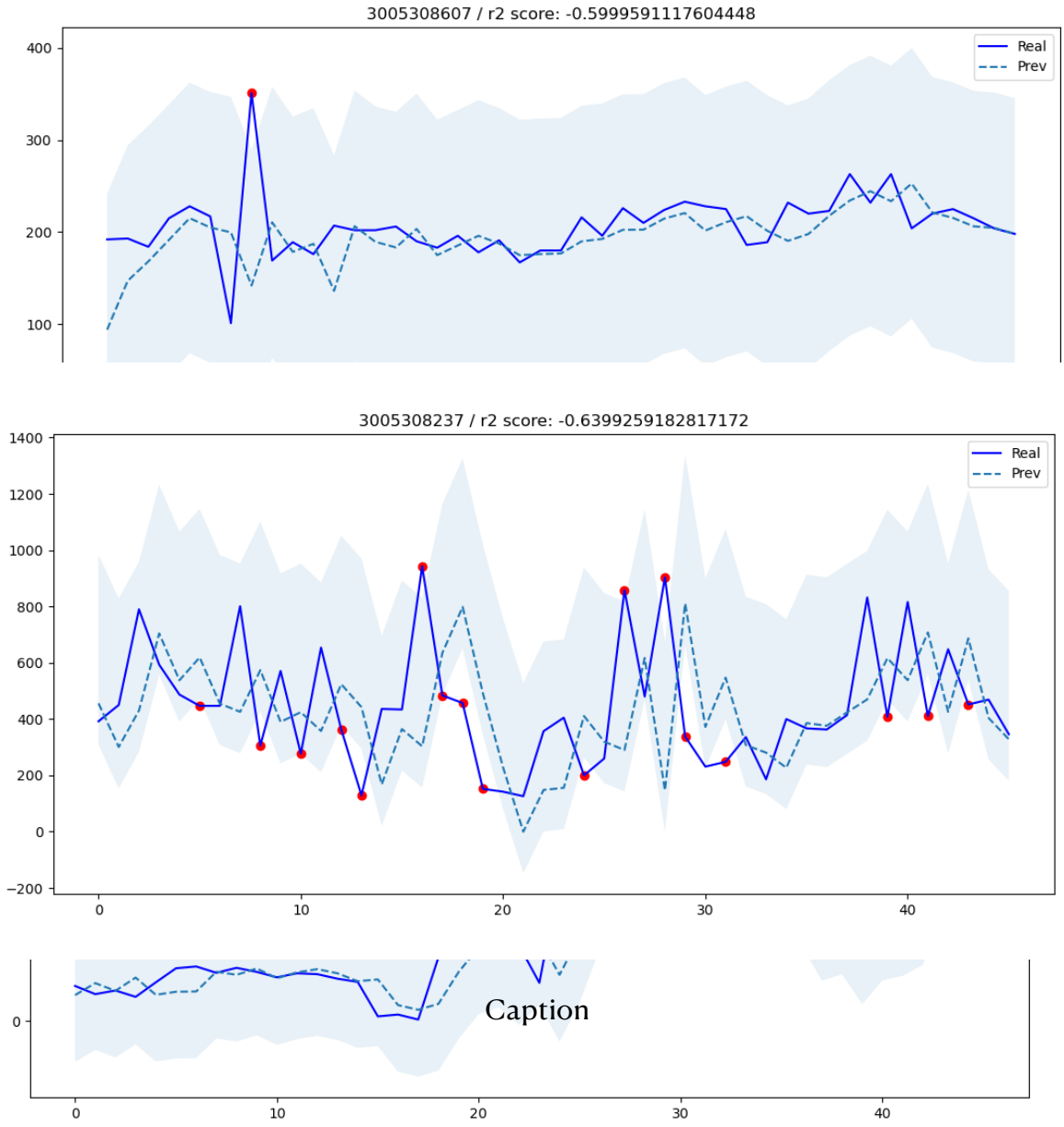

Figura Y - Detecção de múltiplos inícios de vales

As figuras Y e Y mostram a detecção de picos nas curvas que fogem do padrão até então apresentado. A figura Y2 também possui visualmente duas quedas acentuadas que não são detectadas entre os momentos 27 e 35 no eixo X, mesmo que a primeira queda esteja bem próxima do limiar do intervalo padrão. A figura Y3 mostra a detecção na mesma curva de 2 picos em níveis diferentes e um consecutivo decaimento.

A figura Z mostra um exemplo onde o algoritmo detecta diversas novidades em uma curva com uma variância muito grande, com grandes decaimentos seguidas de picos. Este comportamento foi comumente observado em curvas com esta característica e se mostrou como um ponto negativo do modelo treinado.

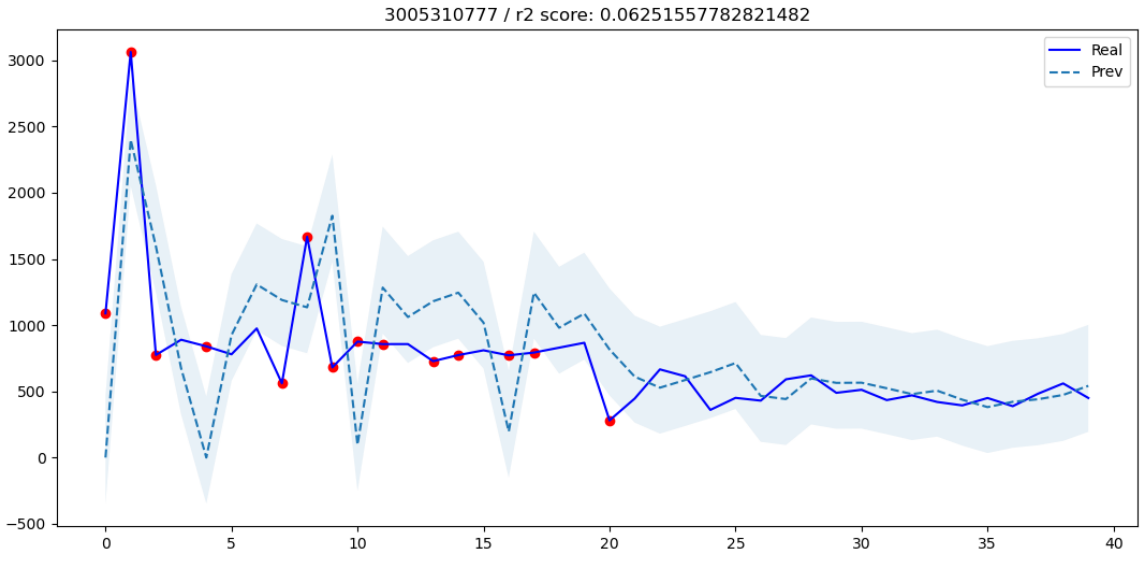

Figura A - Falha na detecção

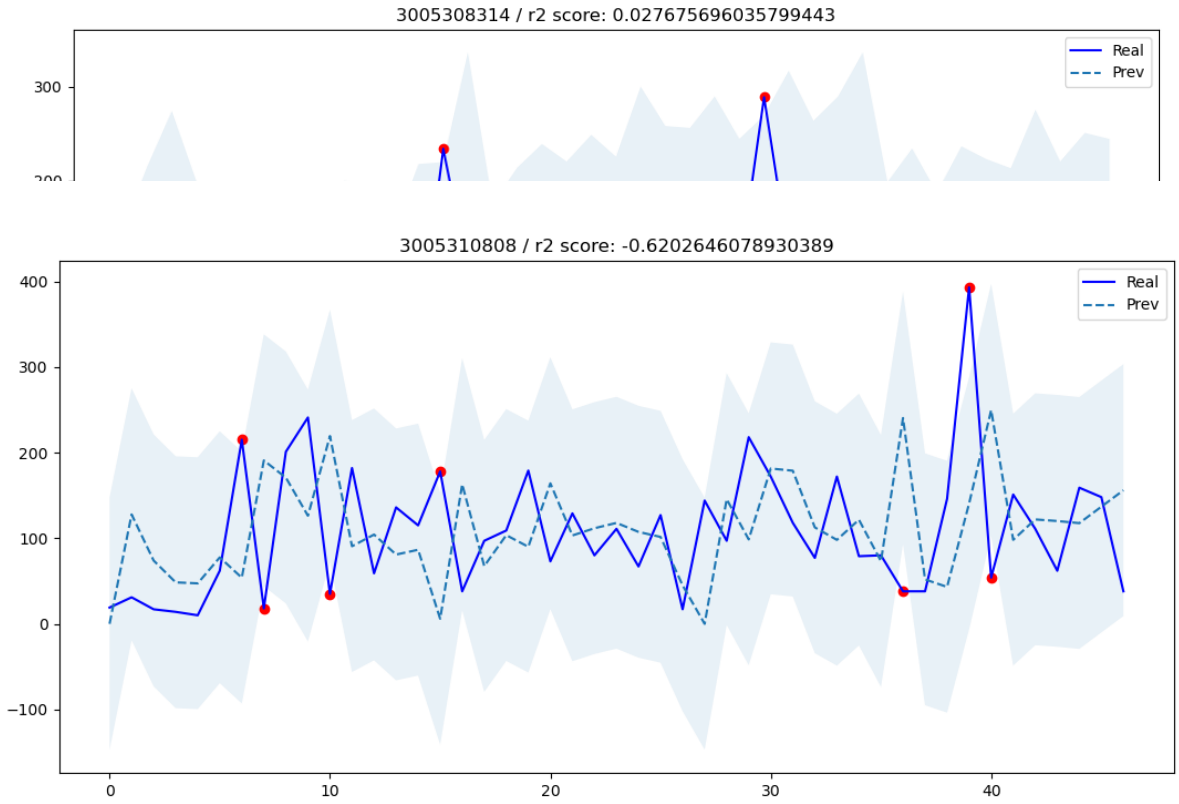

Figura Z - Diversas detecções em curva com alta variância

A figura A demostra um exemplo dos casos que se tentou minimizar com o agrupamento por faixas de consumo médio. Observa-se que o intervalo padrão se vê muito pequeno em relação à figura, isso se deve ao fato dos valores de consumo serem grandes em relação ao intervalo em que ela foi classificada gerando assim um alto nível de detecção provavelmente indevido.

3.3 Performance

A análise de performance é muito importante quando se tem dados massivos em problemas de Big Data e não adiantaria propor uma solução no mercado que não fosse factível com a realidade das empresas. Todos os testes foram realizados em uma máquina Windows 10, 8GB de RAM, Processador Intel i5-7300U 2.60GHz num SSD criptografado, criptografia essa que pode diminuir a performance.

A análise de tempo do serviço de ETL se mostrou bastante razoável. Todo o processo demora cerca de 85 segundos para processar mil clientes ou 0.08 segundos por cliente. Uma das otimizações utilizadas foi a utilização sempre que possível da estrutura de *List Comprehension* do Python. Ela é

normalmente uma forma mais compacta, rápida e elegante de criação de listas[10]. Segue um exemplo de uma List Comprehension:

```
foo = [i if i < 10 else 0 for i in my_list]
```

Este código substitui o formato linear:

```
foo = []
for i in my_list:
    if(i < 10):
        foo.append(i)
    else:
        foo.append(0)
```

Muito mais limpo, claro e muitas vezes mais rápido que o formato linear padrão.

Outra otimização nesta etapa levou em consideração o fato de que a inserção de muitos registros de uma vez gasta menos recursos computacionais do que a inserção de muitos dados separadamente. Testes realizados com o mesmo banco e estrutura mostraram o comportamento mostrado na figura X de inserção de registros por gasto de tempo. Apesar dos ruídos, observa-se claramente uma linearidade com inclinação muito baixa mostrando como é interessante inserir mais registros de uma única vez.

A serviço de ETL teve uma performance muito boa, ele realiza a integração de cerca de 100 mil pontos em 56 segundos, ou 1785 pontos por

[10] https://www.programiz.com/python-programming/list-comprehension

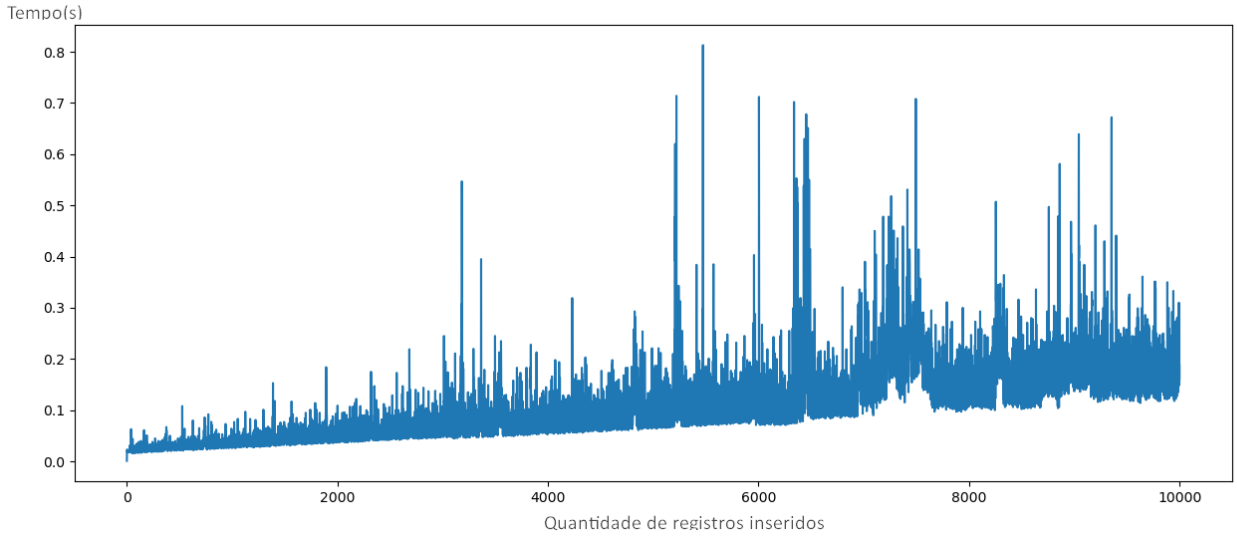

Figura X - Número de Registros por tempo gasto para inserção

segundo, isso é o suficiente para processar dados de quase 30 clientes a cada segundo.

O Árvore de Hoeffding tem uma característica de criação de seus nós e folhas muito rápido, com poucos pontos ela ja consegue adaptar bem sua estrutura. A figura X mostra a evolução da quantidade de Nós, Folhas, Nós ativos e a Profundidade da árvore ao longo do tempo. Neste gráfico o tempo é definido como um modelo salvo após ser treinado incrementalmente a cada 2 milhões de pontos.

Como previsto, rapidamente são criados vários nós e folhas, que permanecem com um crescimento estável ao longo do tempo, a profundidade não varia além de 23 e os nós com aprendizado ativo variam levemente de acordo com a chegada de novos pontos ainda não vistos.

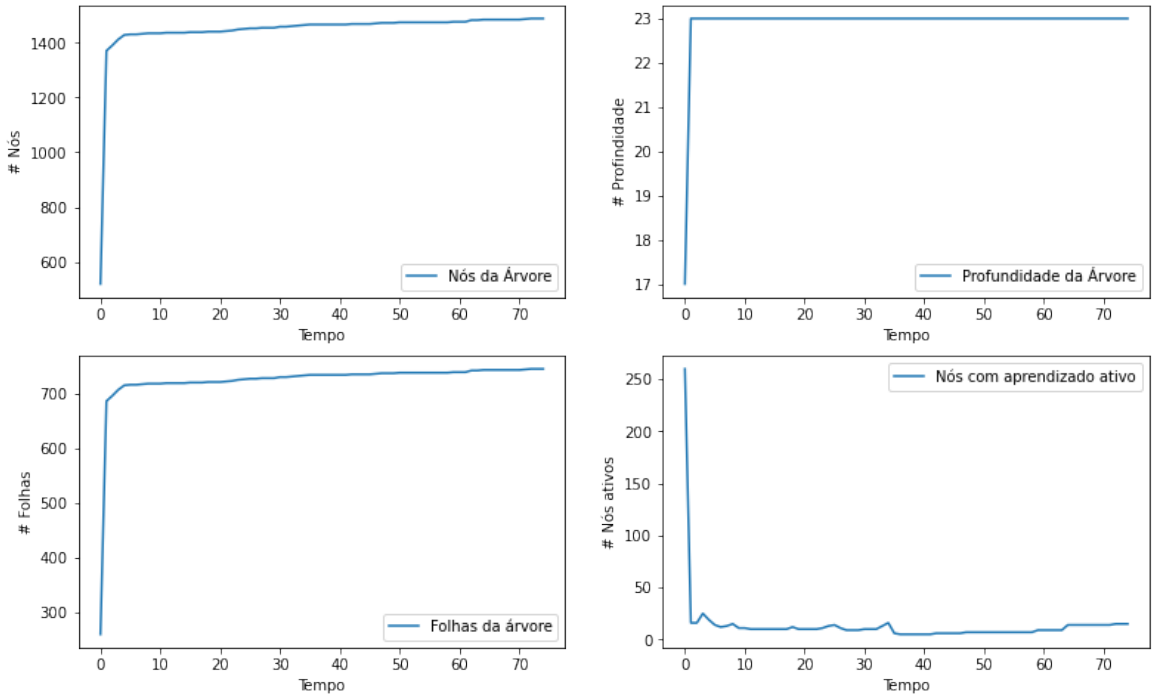

Figura X - Status da Árvore

4 - Lições aprendidas

Entendo que além da descrição da solução desenvolvida durante este tempo é interessante também demonstrar e citar neste trabalho as diversas tentativas e seus resultados parciais ou inconclusivos. Utilizarei nesta sessão uma linguagem informal com intuito de evitar ser prolixo.

4.1 - *Clustering*

No início do desenvolvimento de ideias comecei tentando utilizar algum método de agrupamento de séries temporais. Mas inicialmente não fui muito feliz nessa abordagem. A idéia de classificação em um grupo veio do princípio de detecção de novidade sair de um padrão previamente conhecido e talvez fazer algum tipo de agrupamento ajudaria a encontrar este padrão. A idéia de agrupamento das curvas usando a própria curva

como característica se mostrou falha quanto a encontrar algum algoritmo que fizesse a classificação de uma nova curva sem ter que reclassificar todo o conjunto.

A estratégia de agrupamento voltou a ter destaque no processo quando surgiu o problema de *Big Data*. Desta fez o agrupamento serviria para fazer uma pre-classificação para que se pudesse treinar vários modelos de regressão a fim de eliminar o problema de carregar todos os dados na memória para o treinamento de um modelo único. A figura X ilustra esse processo os dados de treinamento são agrupados em *n* grupos e consequentemente esses grupos são utilizados para para treinar *n* modelos. Um novo dado passa por esse mesmo fluxo, é classificado em um grupo e seria utilizado um dos modelos para a predição deste.

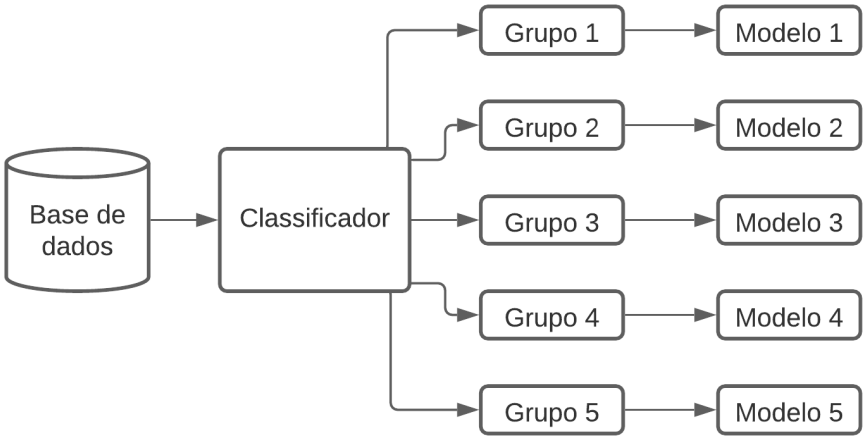

Figura X - Fluxo de Agrupamento

A idéia é muito boa se houvesse outros dados de agrupamento das curvas que não seus próprios valores de consumo como tipo do cliente ou localização do mesmo. Mas da forma que está, faz sentido supor que a grande maioria dos clientes são do tipo residencial/pequeno-medio porte e

tem um padrão de consumo similar. Bem, vamos aos resultados para mostrar

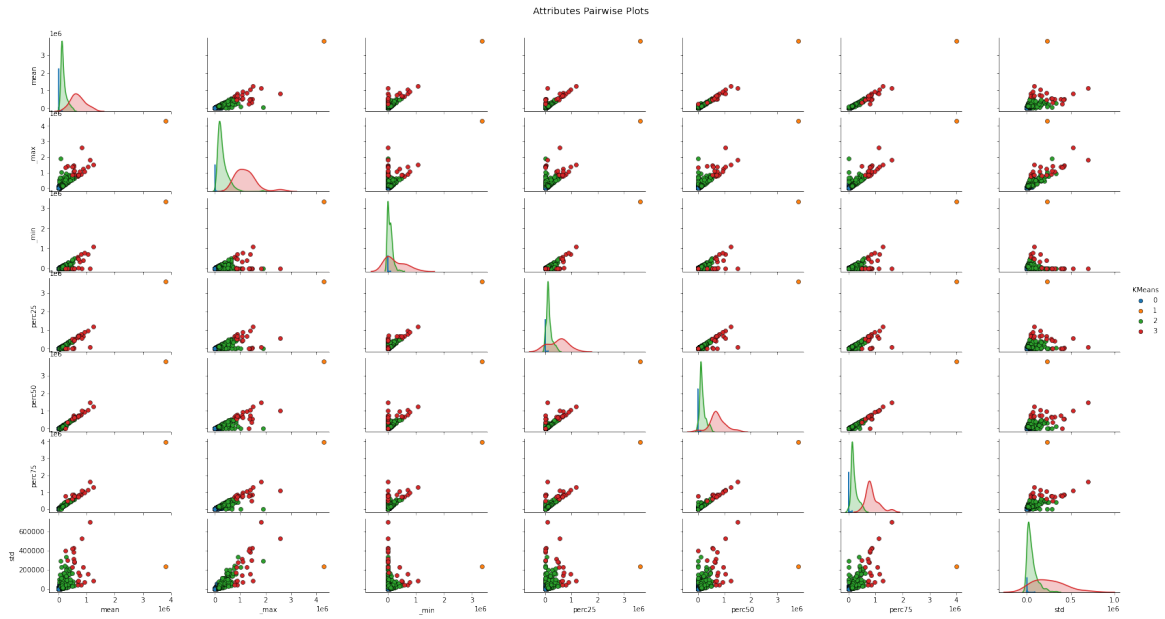

Figura X - Agrupamento 1

porque no nosso caso a idéia não se aplica.

A figura X mostra o resultado do primeiro agrupamento, par-a-par das variáveis utilizadas no agrupamento. As variáveis escolhidas como características da curva foram as características estatísticas dela, como media, mediana, desvio padrão e 25, 50 e 75 percentils. O *KMeans* do pacote *ScikitLearn* foi escolhido devido ao mesmo problema de quantidade de dados. Todos os outros algoritmos de agrupamento do pacote requeriam uma alocação de memória absurda (mais 600 Terabytes em alguns casos) e o *KMeans* é um dos poucos que retorna os clusters de cada grupo e consecutivamente permite que uma nova curva seja classificado.

Percebe-se que mesmo escolhido 4 grupos inicialmente, quase não se nota mais que dois grupos na figura, isso se da pela extrema densidade dos dados em alguns pontos. A Figura X mostra em destaque alguns dos gráficos da figura X onde nota-se 3 grupos sendo o da cor laranja com um único ponto e o verde extremamente denso.

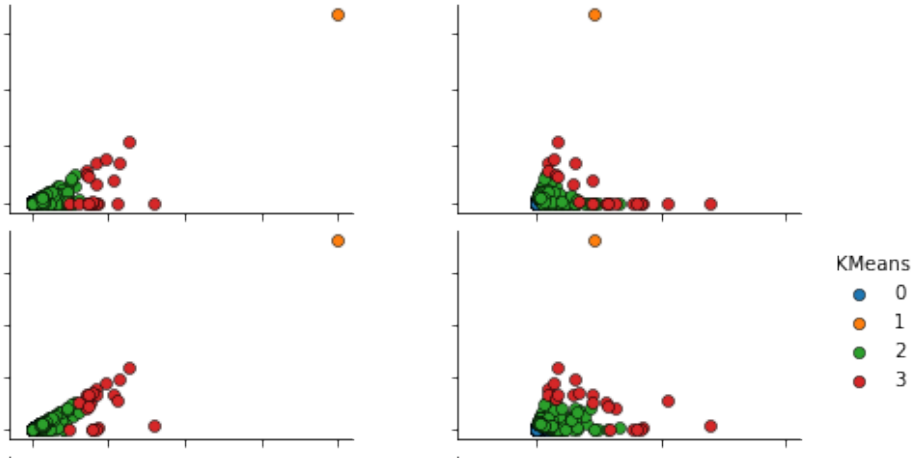

Figura X - Zoom do agrupamento

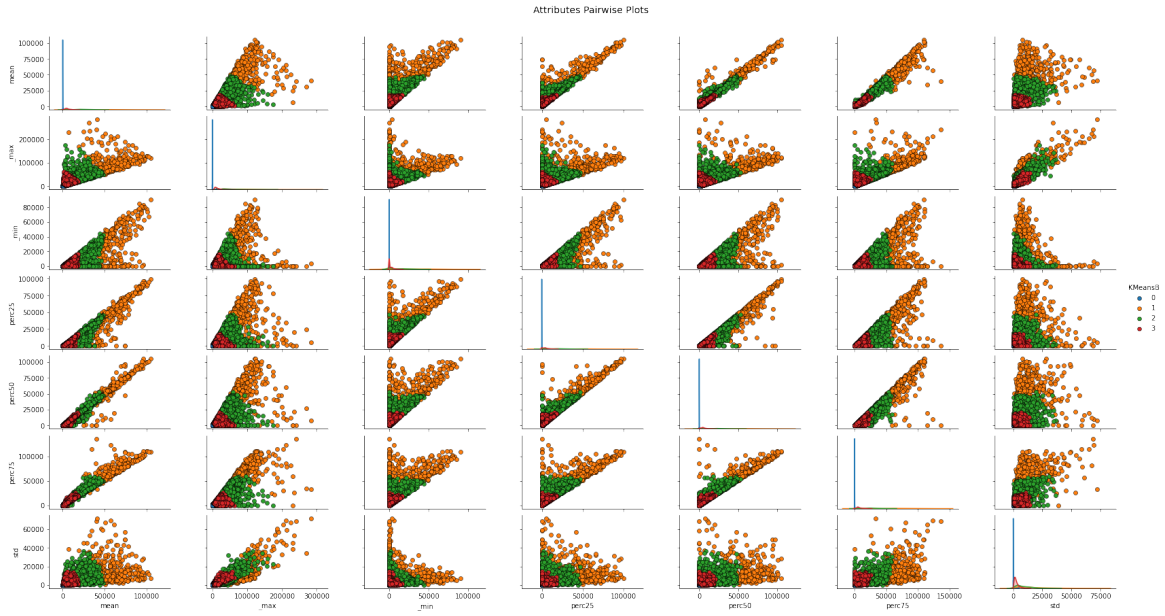

Figura X - Segundo agrupamento

Em números dos 9,5 milhões de pontos iniciais, o grupo maior ficou com 9,37 milhões, o que não justificaria a abordagem. Uma última tentativa foi feita reagrupando o grupo maior e o resultado foi praticamente o mesmo. Pela figura X e Y, parece ter funcionado, mas como pode ser visto no gráfico de dispersão da figura X, o grupo com mais pontos é o azul e está tão aglomerado no na ponta do gráfico que não é possível sequer vê-lo. Dos 9,37 milhões de pontos do grupo gerador deste, o grupo azul continuou com 9,31 milhões confirmando a infactibilidade da abordagem.

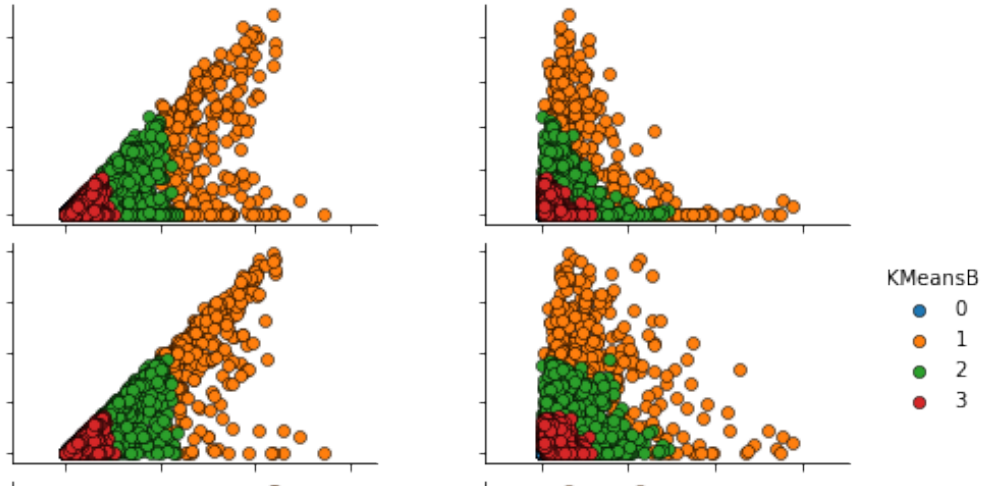

Figura Y - Zoom da figura X

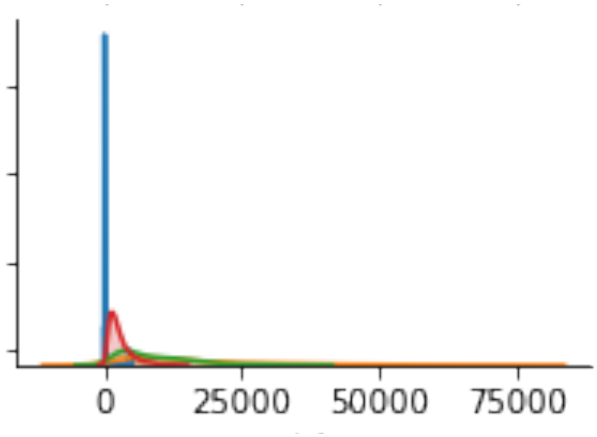

Figura X - Dispersão dos grupos

4.2 - SciKitLearn *Random Forest* e LightGBM

Continuando a pesquisa com outras abordagens, comecei a investir na na predição de time series que como visto nas sessões anteriores foi escolhida como solução final.

Trabalhei com bibliotecas de ORM (SQL Alchemy), salvando os dados processados e dados relevantes pro modelo em um banco de dados (SQLite) o que foi muito útil para os testes. Uma dica valida de citar aqui é: caso tenha uma base muito grande, não perca tempo com transformação de toda sua base antes de ter uma certa certeza do rumo que irá seguir, faça testes incrementais e teste hipóteses para entender qual será possivelmente a melhor abordagem.

Com o propósito de previsão testei alguns métodos como Gradient Booster Regressor e o Random Forest Regressor e obtive algumas especificidades com cada um. O pacote do *Scikitlearn* possui ótimos métodos implementados com diversas otimizações de consumo de tempo e memória, porém ele se mostrou muito aquém das expectativas quando se tratava em treinar com uma base muito grande. Após diversos testes consegui utilizar somente 200.000 pontos para treinamento do *Random Forest*, ou seja, menos de 4 mil curvas do meu dataset de 9,5 milhões. Qualquer quantidade acima deste valor retornava erro de memória.

O *Light Gradient Boosted Machine* ou *LightGBM*[11] do pacote de mesmo nome é uma ótima alternativa pois possui uma velocidade e uso de memória incríveis comparado ao *Random Forest* do *ScikitLearn*, foi possível treinar um modelo com cerca de 5 milhões de pontos (algo próximo de 0.01% da base disponível e 10 vezes mais comparado ao Random Forest) e os resultados foram razoáveis mostrados na Figura X, com o previsto em tracejado e o real nos pontos verdes. A medida de erro percentual SMAPE neste caso deu 0.0794.

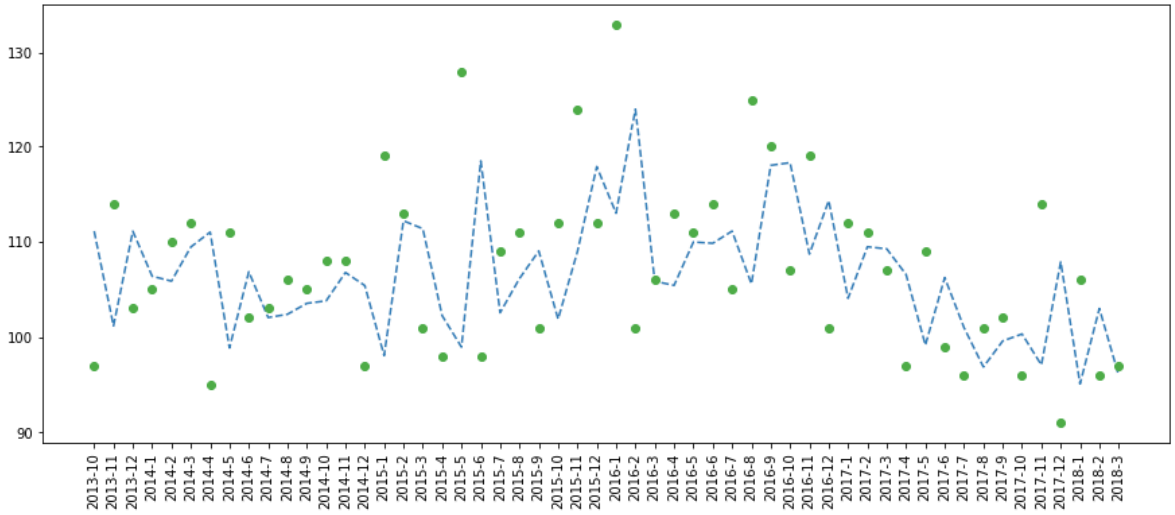

Figura X - Exemplo do LightGBM

Nenhum desses métodos porém, permitia o aprendizado incremental do modelo sem treinar novamente. Pensando numa abordagem numa empresa com novos dados constantes e com a memória RAM limitada, seria necessário um gerenciamento de quais dados seriam utilizados para cada treinamento realizado ao longo do tempo e os resultados poderiam variar muito. Caso o problema tenha dados muito estáveis e temporalmente independente talvez poderia ser uma boa opção, já que uma vez treinado o modelo poderia permanecer fixo.

4.2 - Apache Spark

[11] https://lightgbm.readthedocs.io/en/latest/

Após me deparar com o problema de *Big Data* pesquisei e encontrei algumas soluções interessantes muito utilizadas no mercado, meu foco passou a ser então estudar aprender e testar uma destas que parecia muito promissora: o Apache Spark.

> "Apache Spark é um motor unificado para problemas de análise de processamento de dados de larga escala"[12] - Tradução literal da página oficial

Realizei 3 cursos online sobre machine learning que citavam a ferramenta e mais dois específicos para ela (prática que recomendo bastante), li muitos artigos e mergulhei na documentação. Depois de algumas semanas de trabalho consegui adaptar quase tudo que tinha feito com a abordagem tradicional no ambiente do Spark. O Spark tem alguns conceitos interessantes como transformations '*Lazy*' onde se pode transformar os dados de diversas formas sem ter que realmente carrega-los. O carregamento e o processamento só ocorre quando uma ação é chamada como collect() ou mesmo show(). Possui também o conceito de Pipelines, conhecido de outros frameworks, que seria um caminho ou passo-a-passo por onde os dados passariam por diversas transformações até chegar no modelo final. Ele também possui implementações próprias de modelos mais utilizados para diversos problemas.

O Spark parecia muito promissor, e seu sistema de gerenciamento poderia conseguir resolver a questão da memória de tal forma que ele conseguiria utiliza o disco para salvar os dados assim que a RAM se esgotasse. Não consegui concluir com essa abordagem, somente uma vez após diversas tentativas de configurações diferentes, consegui treinar meu

[12] https://spark.apache.org/

modelo após 9 horas de treinamento (o que é pouco relativo ao tamanho da minha base) e mesmo assim, ele ficou inutilizável apesar de não haver erros nos *logs* e nenhuma interrupção abrupta, não consegui usa-lo nem salva-lo.

Ainda fiz duas últimas tentativas utilizando o Spark para aquilo que ele era realmente feito: Computação em Clusters. Nesse sentido, iniciei um novo curso que ensinava passo a passo como utilizar o Spark na AWS[13]. Configurei 3 clusters de 16Gb de RAM para rodar meu código, fiz alguns testes e falharam. Tentei novamente com o dobro: até 6 Clusters escaláveis de 32Gb de RAM cada um e continuei tendo problemas. Não consegui chegar a uma conclusão a respeito do problema, algumas fontes e fóruns apontavam falta de memória, mas, não encontrei nada conclusivo.

Apesar dos resultados desta abordagem terem sido frustrantes, o aprendizado foi muito grande com tudo isso, a principal motivação foi entender que problemas reais de *Big Data* estão em todos os lugares e o Spark é uma das principais ferramentas para escalar estes problemas e pretendo investir mais algum tempo nela num futuro próximo.

5 - Conclusão

Aprendizado por fluxo contínuo de dados é uma abordagem muito interessante por ter como princípio o carregamento particionado dos dados e a possibilidade de um treinamento incremental. Ela reduziu os custos computacionais e possibilita tratar problemas com quantidades muito grandes de dados.

A detecção de novidade pode ser aplicada em diversos problemas. No caso das perdas não técnicas com dados não categorizados como normal e anormal, é possível chegar a resultados visualmente muito satisfatórios, mas

[13] aws.amazon.com

somente uma inspeção presencial pode constatar realmente uma perda não técnica efetiva.

6 - Referências

[1] https://www.sas.com/pt_br/insights/data-management/o-que-e-etl.html - ETL O que é e qual a sua importância

[2] S. K. Bansal and S. Kagemann, "Integrating Big Data: A Semantic Extract-Transform-Load Framework," in Computer, vol. 48, no. 3, pp. 42-50, Mar. 2015, doi: 10.1109/MC.2015.76.

[3] https://www.mariofilho.com/how-to-predict-multiple-time-series-with-scikit-learn-with-sales-forecasting-example/

[4] PROPOSTA DE UM ALGORITMO GENERICO ´ DE DETEC ¸CAO DE NOVIDADES EM S ˜ ERIES ´ TEMPORAIS UTILIZANDO MODELOS DE PREVISAO - 17 de julho de 2007 - ANDRE PAIM LEMOS

[5] A regression tree approach using mathematical programming LingjianYang et al - Expert Systems with Applications Volume 78, 15 July 2017, Pages 347-357

[6] https://towardsdatascience.com/tree-based-methods-regression-trees-4ee5d8db9fe9 - Kushal Vala - Tree-Based Methods: Regression Trees

[7] Gomes, Heitor Murilo & Read, Jesse & Bifet, Albert & Barddal, Jean Paul & Gama, Joao. (2019). Machine learning for streaming data: state of the art, challenges, and opportunities. ACM SIGKDD Explorations Newsletter. 21. 6-22. 10.1145/3373464.3373470.

[8] A Survey on Hoeffding Tree Stream Data Classification Algorithms - Arvind Kummar, Parminder Kaur and ratibha Sharma CPUH-Research Journal: 2015, 1(2), 28-32

[9] Performance analysis of Hoeffding trees in data streams by using massive online analysis framework, P.K. Srimani,Bangalore University Int. J. Data Mining, Modelling and Management, Vol. 7, No. 4, 2015

[10] A Survey on Hoeffding Tree Stream Data Classification Algorithms - Arvind Kummar, Parminder Kaur and ratibha Sharma CPUH-Research Journal: 2015, 1(2), 28-32

[11] - Hoeffding Decision Trees http://huawei-noah.github.io/streamDM/docs/HDT.html

[12] Aguayo, Leonardo & Barreto, Guilherme. (2017). Novelty Detection in Time Series Using Self-Organizing Neural Networks: A Comprehensive Evaluation. Neural Processing Letters. 10.1007/s11063-017-9679-2.