Capítulo

1

Processamento e Análise de *Big Data* e Aplicação de Algoritmos de *Machine Learning* através da utilização da Plataforma *HPCC Systems*

Mauro Donato Marques (LexisNexis Risk Solutions) e Alysson Rogerio Oliveira (LexisNexis Risk Solutions)

Abstract

Throughout the course, participants will have the opportunity to learn the essential concepts of massive data volume processing and analyzing (Big Data) and the process of developing a query service using the open-source platform High-Performance Computing Cluster (HPCC Systems) and also the use of Machine Learning algorithms, as well as having the possibility to apply the knowledge acquired in a training environment available in the classroom.

Resumo

Ao longo do minicurso, os participantes terão a oportunidade de conhecer os conceitos essenciais de processamento e análise de volumes massivos de dados (Big Data) e o processo de desenvolvimento de um serviço de consulta fazendo uso da plataforma open-source composta por um Cluster Computacional de Alto Desempenho (HPCC Systems) e, também, a utilização de algoritmos de Aprendizado de Máquina, bem como terão a possibilidade de aplicar os conhecimentos adquiridos em um ambiente de treinamento disponibilizado em sala de aula.

Informações Técnicas: Curso de nível básico. O curso requer apenas um computador com acesso à Internet e uma conta no GitHub.

GitHub repositório: https://github.com/mauromarx/WSCAD_2023

1.1. A Plataforma HPCC Systems

HPCC Systems (High Performance Computing Cluster) é uma plataforma para solução de desafios de Big Data com as seguintes características:

- Supercomputação: processamento paralelo e dados distribuídos;
- Open source: código aberto e gratuita;
- Completa: gestão compreensiva e simplificada do fluxo de dados.

HPCC Systems oferece o melhor de ambos os mundos no que tange ao processamento e análise de volumes massivos de dados (Big Data), ou seja, combina o rápido desempenho de um "Data Warehouse" para a entrega de informações com a capacidade de tratar os dados como se estivessem em um "Data Lake". HPCC Systems usa arquitetura de dados distribuída e uma metodologia de processamento paralelo para trabalhar com grandes conjuntos de dados.

1.1.1. Um breve histórico

A plataforma de Big Data que se tornaria HPCC Systems foi desenvolvida em 2001 por uma equipe interna de engenharia da LexisNexis Risk Solutions. Os primórdios da base tecnológica para a plataforma HPCC Systems foi desenvolvida pela primeira vez na Seisint em 1999, a fim de gerenciar um número significativo de conjuntos de dados. Em 2000, a equipe da Seisint precisava coletar e analisar grandes quantidades de informações brutas de consumo de dados financeiros de uma ampla variedade de fontes para um cliente responsável por determinação da "score" de crédito ao consumidor nos Estados Unidos. Isto levou à criação da linguagem de programação, conhecida como ECL (Enterprise Control Language), que HPCC Systems usa atualmente. Em 2004, a LexisNexis Risk Solutions (LNRS) adquiriu Seisint junto com sua tecnologia, incluindo a linguagem ECL usada para programação em Clusters Thor e Roxie. Durante esse período, a plataforma HPCC Systems era empregada, principalmente, como uma ferramenta interna da LexisNexis Risk Solutions, e ainda não havia atingido o seu pleno potencial. Em 2008, a LexisNexis Risk Solutions adquiriu a ChoicePoint, uma seguradora e provedora de análise e, nos próximos três anos, o portfólio de produtos da ChoicePoint foi integrado à plataforma HPCC Systems. Combinando a plataforma HPCC Systems com os produtos de seguros ChoicePoint criou-se negócios poderosos e vantajosos - uma solução de processamento de Big Data extremamente eficiente, capaz de gerenciar grandes quantidades de dados, gerando produtos de tratamento de dados muito mais eficientes no já volumoso mercado de seguros.

Em 2011, a LexisNexis decidiu lançar a plataforma HPCC Systems sob uma licença de código aberto. Desde o seu lançamento, a HPCC Systems desenvolveu uma rica comunidade de usuários e uma rede global de clientes. Além disso, a HPCC Systems continua a inovar a plataforma, adicionando suporte para as arquiteturas baseadas em nuvem; em desenvolvimento de novas ferramentas de administração, governança e orquestração de dados; e adicionando novos recursos de dashboard.

Os usuários atuais da plataforma HPCC Systems que podem ser mencionados publicamente incluem a Quod, um Bureau de Crédito no Brasil, o qual a LNRS ajudou a criar, e muitas universidades proeminentes, como: Clemson University, Florida Atlantic University, Kennesaw State University, RV College of Engineering, Universidade de São

Paulo, Universidade Federal de Santa Catarina e várias outras universidades em todo o mundo.

Figura 1.1: A evolução da plataforma HPCC Systems.

1.1.2. O que é a plataforma HPCC Systems?

HPCC Systems é uma plataforma de "Data Lake" de código aberto (open source) projetada para obter dados de diversas fontes de dados em formatos estruturados e não estruturados. Os dados geralmente são armazenados em arquivos simples, como arquivos básicos de disco, ou em armazenamentos de objetos como Amazon S3 e armazenamento BLOB do Azure. Em outras palavras, não há um esquema de formatação pré-definida, na qual os dados precisam ser ajustados antes do armazenamento.

A implementação de uma típica plataforma HPCC Systems começa com apenas algumas fontes de dados, alguns processos de análises iniciais e ferramentas de relatório, mas o tamanho, a complexidade e a capacidade do "Data Lake" pode crescer rapidamente. Depois que os dados são ingeridos no "Data Lake" e, refinados através de limpeza e padronização dos dados, começa o processo de enriquecimento desses dados. Esse enriquecimento de dados é um processo iterativo e evolutivo que extrai tanto conhecimento quanto possível das fontes de dados. Uma vez que esse conhecimento é extraído, ele fica disponível para outros usuários do "Data Lake", que precisam dos mesmos por meio de um processo conhecido como entrega de dados. Durante a entrega de dados, a plataforma HPCC Systems garante que os dados sejam transferidos aos usuários do "Data Lake" de maneira responsiva e segura. Uma analogia pode ajudar a ilustrar o que acontece com os dados à medida que eles entram em um sistema de "Data Lake" do HPCC Systems. Nesta analogia, a água (dados brutos) é coletada em um reservatório (Data Lake) onde em seguida, é processado para torná-lo adequado ao consumo do público.

Para continuar entregando água à população, a usina de beneficiamento não pode parar a coleta ou processamento de água; o processo deve fornecer água de forma confiável 24 horas por dia, sete dias por semana para acompanhar a demanda do consumidor. Um "Data Lake" deve oferecer o mesmo nível de disponibilidade. Não importa quantos dados sejam adicionados ao sistema, o processo de catalogação e análise desses dados deve operar continuamente em níveis de serviço de "cinco noves" (o "Data Lake" deve estar ativo e operacional pelo menos 99,999 por cento do tempo).

A figura abaixo registra o ciclo de vida dos dados em um sistema real de “Data Lake” do HPCC Systems atualmente em uso por um cliente da plataforma. Movendo da esquerda para a direita, as fontes de dados entregam os dados ao “Data Lake” do HPCC Systems para ingestão, refinamento, enriquecimento, indexação e análise. O HPCC Systems pode gerar relatórios ou dashboards sobre os dados em qualquer etapa do processo, dependendo de qual informação o consumidor necessita. Todos esses processos ocorrem dentro do ambiente de “Data Lake” para produzir resultados consistentes.

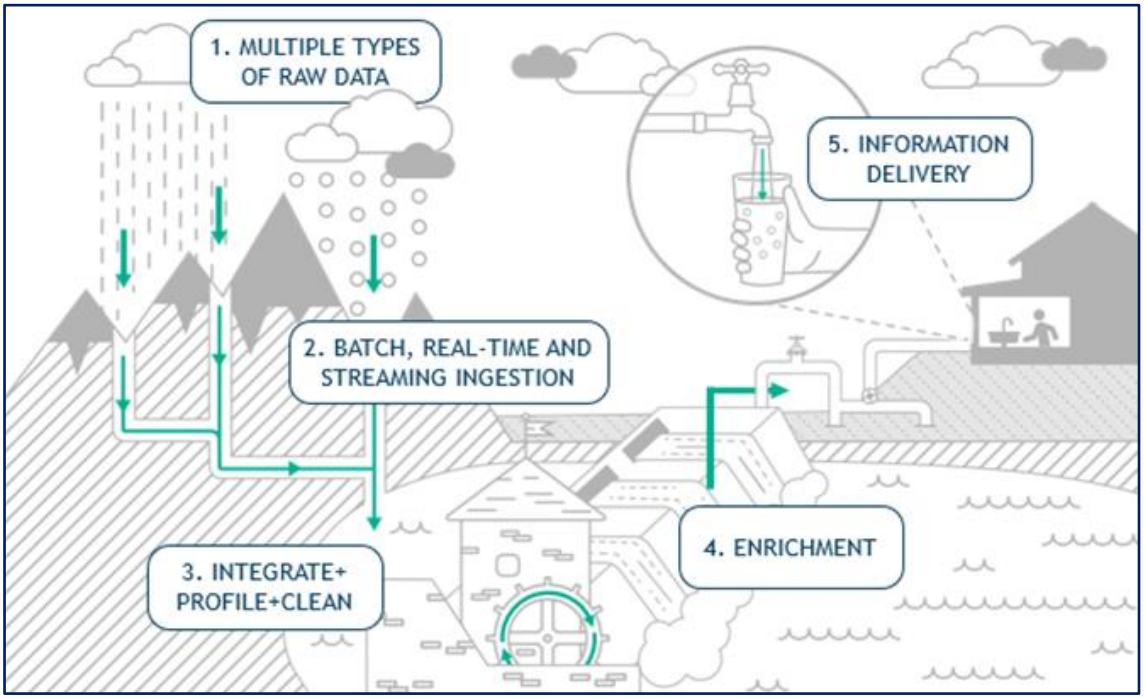


Figura 1.2: Um “Data Lake” deve ser capaz de ingerir, formatar e enriquecer dados com disponibilidade 24 horas por dia, 7 dias por semana.

1.1.3. O Pipeline de enriquecimento de dados no HPCC Systems

O pipeline de dados no HPCC Systems segue os dados desde a origem até sua ingestão no cluster do HPCC Systems, onde é formatado, enriquecido e depois disponibilizado para aplicações hospedadas no Cluster.

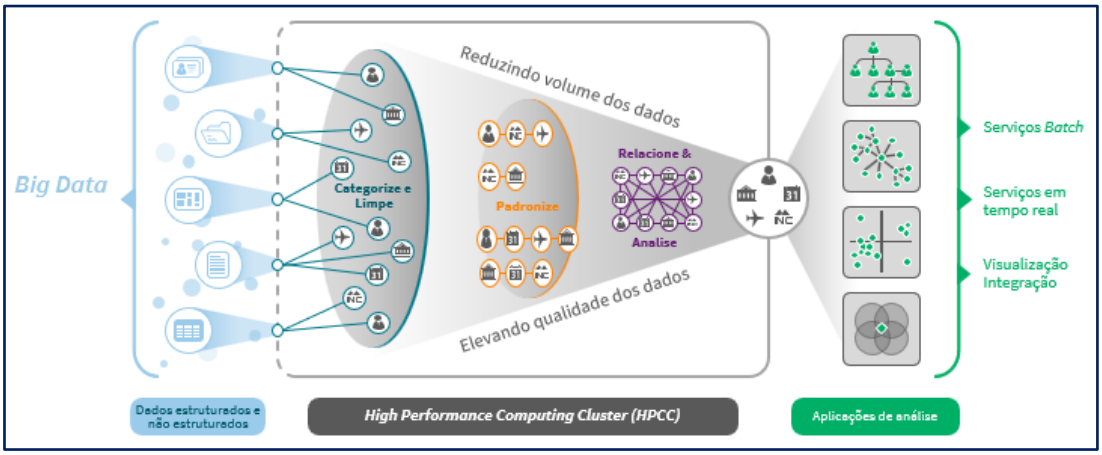


Figura 1.3: Fluxo de dados no HPCC Systems – “Funil” de dados.

1.1.4. Os componentes da plataforma HPCC Systems

O HPCC Systems é composto pelos seguintes componentes:

- A linguagem de programação ECL (Enterprise Control Language) é uma linguagem de programação declarativa e orientada a dados desenvolvida para uso em "Data Lakes" na plataforma HPCC Systems;
- Thor é um cluster de processamento de dados em massa, o qual limpa, padroniza e indexa dados de entrada para uso pelo "Data Lake". Depois que os dados forem refinados pelo Thor, podem, então, serem usados pelo cluster Roxie;
- Roxie é um cluster de API/consulta (query) em tempo real para consultar dados após refinamento por Thor. As consultas Roxie são executadas em menos de um segundo e fornecem resultados de forma concorrente.

Os clusters Thor e Roxie apresentam objetivos específicos e, assim, fazendo uma analogia simplista com o mundo oceânico seria algo como mostrado na figura abaixo:

Figura 1.4: Objetivos dos clusters Thor e Roxie.

1.1.5. A plataforma HPCC Systems

Um diagrama da plataforma HPCC Systems apresentando um cluster Thor (para processamento de dados em massa) e um cluster Roxie (para lidar com consultas de dados) e, ainda, o chamado Power Trio:

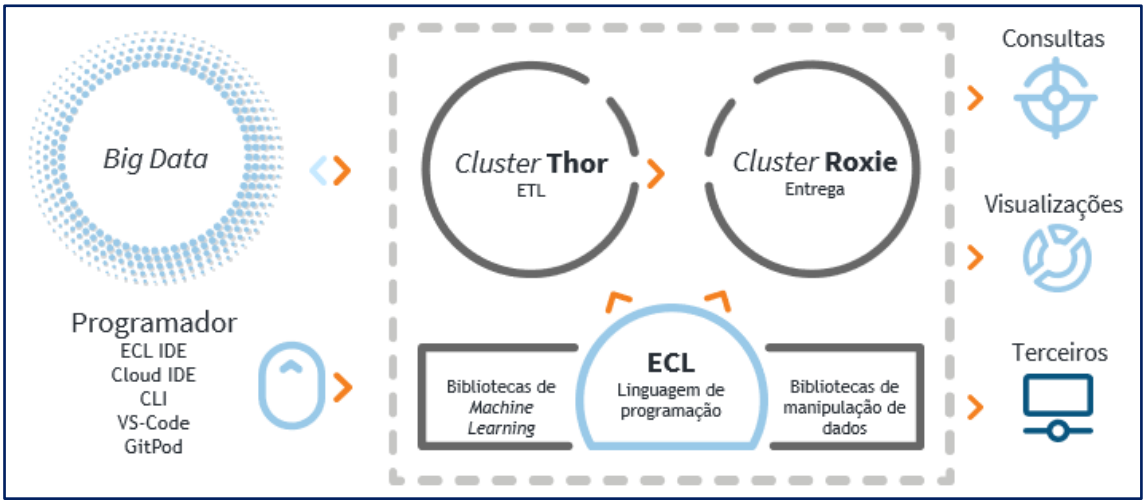


Figura 1.5: Clusters Thor e Roxie.

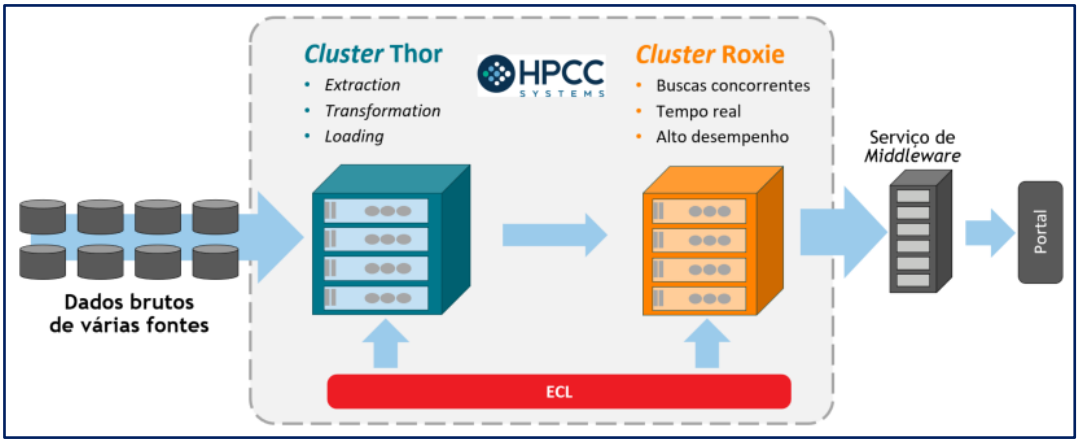


Figura 1.6: Power Trio: A) Thor, B) Roxie e C) ECL.

A) Thor: Arquitetura

Thor é um cluster projetado especificamente para executar processos de manipulação massiva de dados (ETL). Thor é um cluster de preparação de dados de back-office e não se destina a consultas de nível de produção do usuário final.

Os clusters Thor são usados para fazer todo o trabalho "pesado" de preparação de dados para processar dados brutos em formatos padrão. Uma vez concluído o processo, os usuários finais podem consultar esses dados padronizados para coletar informações reais.

No entanto, os usuários finais geralmente desejam ver os seus resultados “imediatamente” e, ainda, geralmente mais de um usuário final deseja obter seus resultados ao mesmo tempo. O cluster Thor só funciona em uma consulta por vez, o que o torna inviável para o usuário final, por isso foi criado o cluster Roxie.

B) Roxie: Arquitetura

Roxie é um cluster projetado especificamente para atender as consultas padrão, fornecendo uma taxa de transferência de mais de mil respostas por segundo (a taxa de resposta real para qualquer consulta, logicamente, depende de sua complexidade). Roxie é um cluster de nível de produção projetado para aplicações de missão crítica.

Os clusters Roxie podem lidar com milhares de usuários finais simultâneos e fornecer a todos eles a percepção dos resultados "imediatamente". Ele faz isso permitindo apenas que os usuários finais executem consultas padrão pré-compiladas que foram desenvolvidas especificamente para uso do usuário final no cluster Roxie.

Normalmente, essas consultas usam índices e, portanto, fornecem desempenho extremamente rápido. No entanto, o cluster Roxie é inviável para uso como ferramenta de desenvolvimento, pois todas as suas consultas devem ser pré-compiladas e os dados utilizados devem ter sido implantados anteriormente.

C) A linguagem ECL

ECL é uma linguagem de programação declarativa, que apresenta inúmeras vantagens sobre o modelo de programação mais convencional. A programação declarativa permite ao programador expressar a lógica de uma computação sem descrever seu controle de

fluxo. Em termos leigos, a linguagem ECL permite que os desenvolvedores digam ao sistema o que eles precisam, mas deixa para o sistema determinar a melhor maneira para fazer isso.

Além de simplificar o design e a implementação de algoritmos complexos, também melhora a qualidade do código, minimizando ou eliminando efeitos colaterais no código do "Data Lake", o que facilita o teste de código e simplifica a manutenção do código. O código ECL é mais fácil de entender e verificar, mesmo por pessoas que não estão familiarizadas com o design original, o que ajuda encurtar a curva de aprendizado para novos programadores.

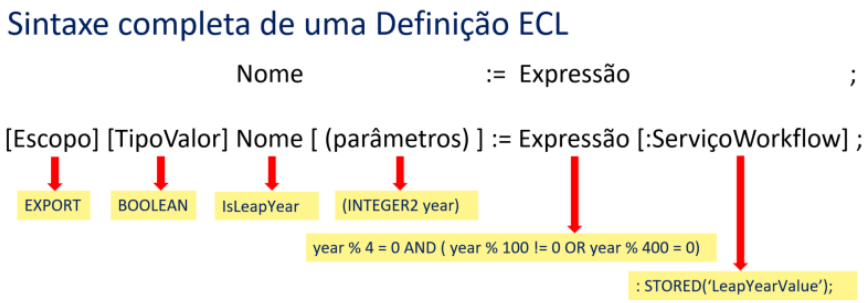
ECL é a única linguagem necessária para expressar algoritmos de dados em toda a plataforma HPCC Systems. No Thor, a linguagem ECL expressa "workflows" de dados que consistem em carregamento dados, transformação, vinculação, indexação etc. No Roxie, a linguagem ECL define consultas de dados. Isso significa que os analistas de dados e programadores da plataforma HPCC Systems só precisam aprender uma linguagem para definir o ciclo de vida completo dos dados.

A linguagem ECL é implicitamente paralela, portanto, o mesmo código ECL desenvolvido para ser executado em um cluster de um nó pode ser executado com a mesma facilidade em um cluster com milhares de nós. O programador não precisa se preocupar em implementar a paralelização, e a linguagem ECL possui uma função otimizadora que garante o melhor desempenho para uma arquitetura específica.

A linguagem ECL foi projetada desde o início para ser uma linguagem de programação orientada a dados. Ao contrário de outras linguagens, funções primitivas de alto nível para dados, como JOIN, TRANSFORM, PROJECT, SORT, DISTRIBUTE, MAP, NORMALIZE etc.; são funções de primeira classe, então operações básicas de dados podem ser implementadas em uma única linha de código. Isto torna a linguagem ECL uma linguagem de programação ideal para análise de dados, pois pode ser usada para expressar algoritmos de dados diretamente, eliminando a necessidade de escrever as especificações do software. Em essência, com o uso da linguagem ECL, os "Data Lakes" no HPCC Systems precisam de menos programadores para entregar mais projetos em menor quantidade de tempo.

- Conceitos básicos de ECL
  - Paradigma declarativo (não-procedural);
  - ECL "não" é sensível a caixa alta/baixa (uppercase/lowercase)
  - Espaço em branco é ignorado para uma melhor leitura;
  - Comentários em linha (//) e em bloco (/* e */);
  - ECL utiliza sintaxe Objeto.Propriedade:
    - Dataset.Campo // um campo em um dataset
    - NomedoDiretorio.Definicao // uma definição em outro módulo
- Ações vs. Definições
  O código ECL é constituído de "Definições" e "Ações".
  - Definições estabelecem o que as coisas são (arquivos de definição ECL):
    - MyDef := 'Hello World'; // não inicia uma WorkUnit

- Ações em ECL resultam em compilação e execução (arquivos BWR):
  - OUTPUT(MyDef); // inicia uma WorkUnit
  - OUTPUT('Hello World, again...'); // inicia uma WorkUnit



1.1.6. Outras ferramentas da plataforma HPCC Systems

A plataforma HPCC Systems fornece para os desenvolvedores um ambiente de desenvolvimento integrado (IDE) denominado como "ECL IDE", a fim de facilitar o desenvolvimento de código ECL. O "ECL IDE" é uma aplicação para o sistema operacional Windows. Há também uma extensão de linguagem ECL disponível para VS Code que alguns desenvolvedores preferem usar.

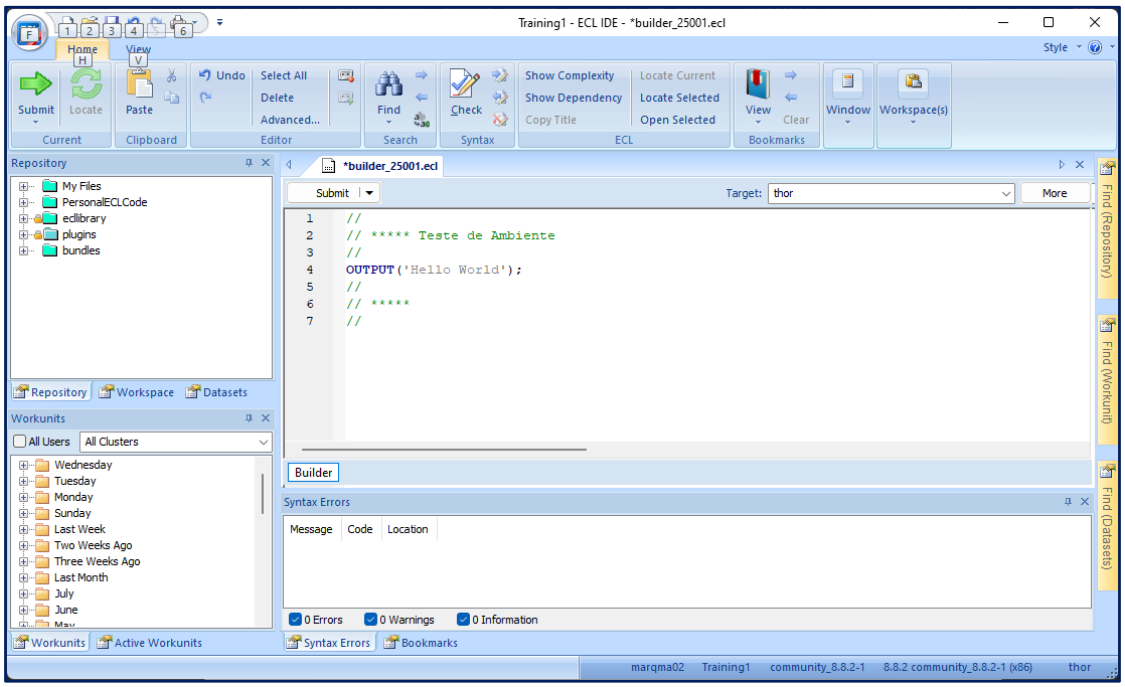


Figura 1.7: ECL Integrated Development Environment (IDE).

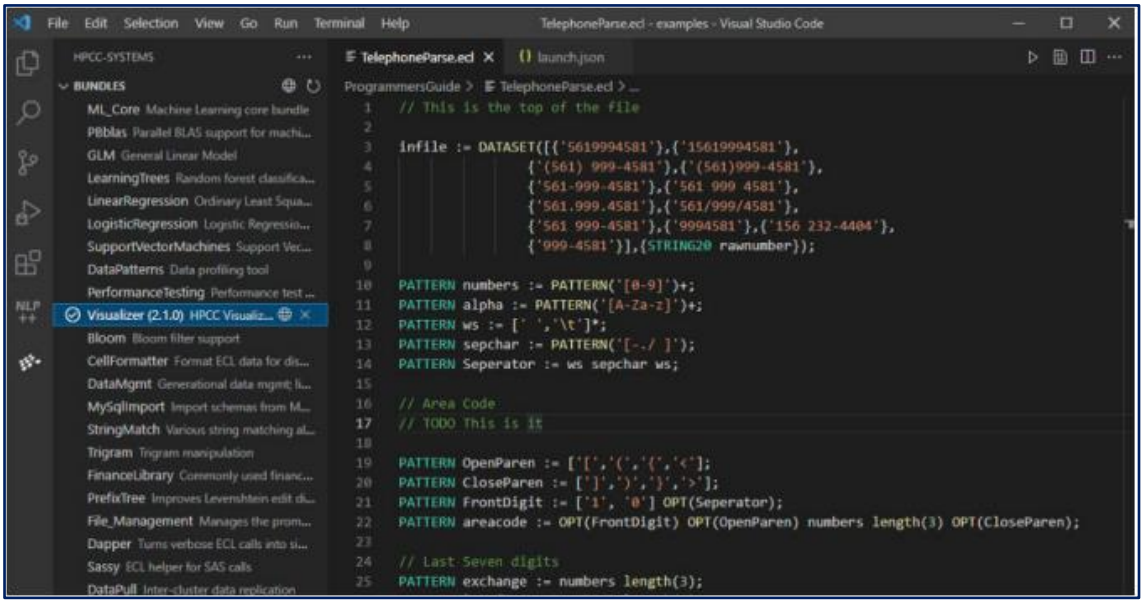


Figura 1.8: VS Code IDE.

1.2. Algoritmos de Aprendizado de Máquina para a plataforma HPCC Systems

A característica principal do Aprendizado de Máquina (Machine Learning) é a capacidade de inferir sobre relacionamentos e, visa prever uma resposta razoável quando apresentado com dados nunca vistos.

O "aprendizado" do Machine Learning (ML) tem várias categorias:

- Supervisionado - O tipo mais comum de ML. Esse método envolve o treinamento do sistema em que os recordsets, juntamente com o padrão de saída de destino, são fornecidos ao sistema para executar uma tarefa;
- Não Supervisionado - Este método não envolve a saída de destino, o que significa que nenhum treinamento é fornecido ao sistema. O sistema precisa aprender por meio da determinação e adaptação de acordo com as características estruturais nos padrões de entrada.
- Deep Learning - Move-se para a área dos métodos ML de Redes Neurais (Neural Networks - NNs). O Deep Learning implica várias camadas maiores que '2' e, também, implica em técnicas utilizadas com dados complexos, como análise de vídeo ou áudio.

1.2.1. Aprendizado Supervisionado

Essa será a categoria abordada nesse estudo com foco no método de Árvores de Decisão (Learning Trees - Random Forests).

A premissa básica do ML Supervisionado é que, dado um conjunto de "amostras de dados" (registros) e um conjunto de "valores-alvo" em campo e formato de registro, que ele aprenda como prever "valores-alvo para novas amostras".

- Amostras de dados: conhecidas como variáveis "Independentes", porque são as informações fornecidas e não dependem de nenhum outro dado. Variáveis Independentes também são conhecidas como 'Características' (features) dos dados.

- Valores-alvo: conhecidos como variáveis “Dependentes”, porque eles são de alguma forma dependentes das amostras de dados.

As variáveis ‘Independentes’ e ‘Dependentes’ juntas são conhecidas como “conjunto de treinamento”.

Dois tipos básicos de Modelos de Análise em Aprendizado Supervisionado:

- Quantitativo - conhecido como “Regressão” - implica um valor numérico;
- Qualitativo - conhecido como “Classificação” - implica uma categoria ou, às vezes, um resultado binário.

1.3. Machine Learning bundles da plataforma HPCC Systems

Os bundles de produção (excluindo os bundles de suporte ML_Core e PBblas) fornecem uma interface principal muito semelhante para o Machine Learning. No entanto, cada algoritmo tem suas próprias peculiaridades (suposições e restrições) que devem ser levadas em consideração. Portanto, é importante ler a documentação que acompanha cada bundle para usá-lo efetivamente.

➢ Link definitivo para todos os bundles de produção, sua documentação e tutoriais, quando aplicável:
https://hpccsystems.com/download/free-modules/machine-learning-library

a) Bundles Principais:

- ML_Core - Machine Learning Core
  Fornece as principais definições de dados para ML. É um pré-requisito para todos os outros pacotes configuráveis de produção.
  Mais informações: https://github.com/hpcc-systems/ML_Core

- PBblas - Parallel Block Basic Linear Algebra Subsystem
  Fornece operações de matriz escalonáveis e distribuídas usadas por vários dos outros pacotes configuráveis. Também pode ser usado diretamente sempre que as operações da matriz estiverem em ordem. Essa é uma dependência para vários dos outros pacotes configuráveis.
  Mais informações: https://github.com/hpcc-systems/PBblas

b) Bundles de Aprendizado Supervisionado:

- LearningTrees (baseado no algoritmo clássico “ML RandomForest”)
  Classificação e Regressão baseadas em Árvores de Decisão. Um dos melhores métodos de ML "prontos para uso", pois faz poucas suposições sobre a natureza dos dados e é resistente ao “overfitting” (*). Capaz de lidar com muitas variáveis independentes. Cria uma “floresta” de diversas Árvores de Decisão e calcula a média das respostas das diferentes Árvores.
  Mais informações: https://github.com/hpcc-systems/LearningTrees

(*) Superajuste (overfitting): Muitos algoritmos tendem a superestimar o conjunto de treinamento. Isso significa que ele está reduzindo o erro de previsão ajustando-se ao ruído que ocorre nesse conjunto de dados. Um modelo de excesso de ajuste tratará esse ruído como sinal e usará o que aprendeu para prever os próximos dados apresentados. Infelizmente, esse próximo conjunto de dados estará sujeito a um conjunto de ruído completamente diferente, o qe não fornecerá bons resultados.

1.4. ML Supervisionado: Árvores de Decisão (Learning Trees - Random Forests)

As Árvores de Decisão têm sido utilizadas, pelo menos, desde a década de 1930 como uma forma de estruturar o conhecimento usando um conjunto de regras em cascata. Elas são conceitualmente simples e razoavelmente fáceis de entender e interpretar.

O LearningTrees bundle fornece uma implementação eficiente e escalável dos métodos Learning Trees. Atualmente, fornece algoritmos de "Decision Trees", "Random Forest", "Gradient Boosted Trees" e "Boosted Forest".

Diante dos diversos algoritmos disponíveis, qual algoritmo se deve escolher?

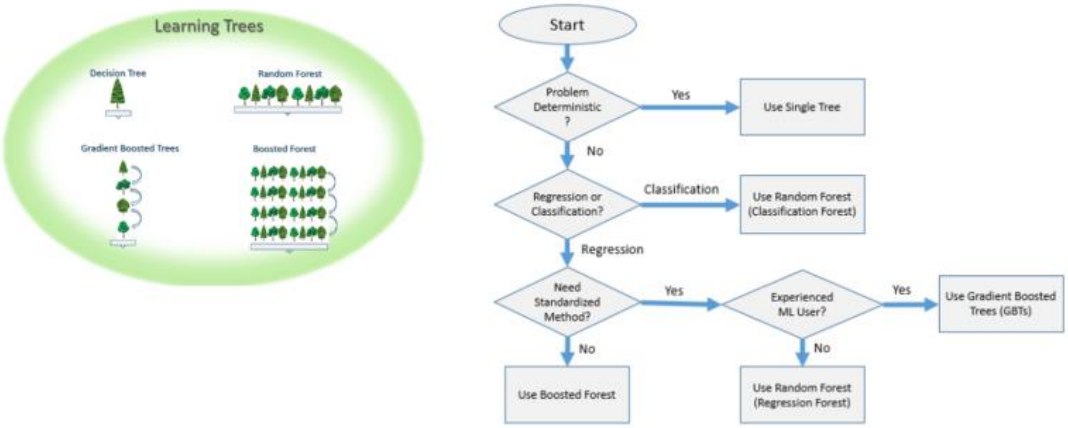


Figura 1.9: Fluxograma a ser usado para auxiliar nessa escolha.

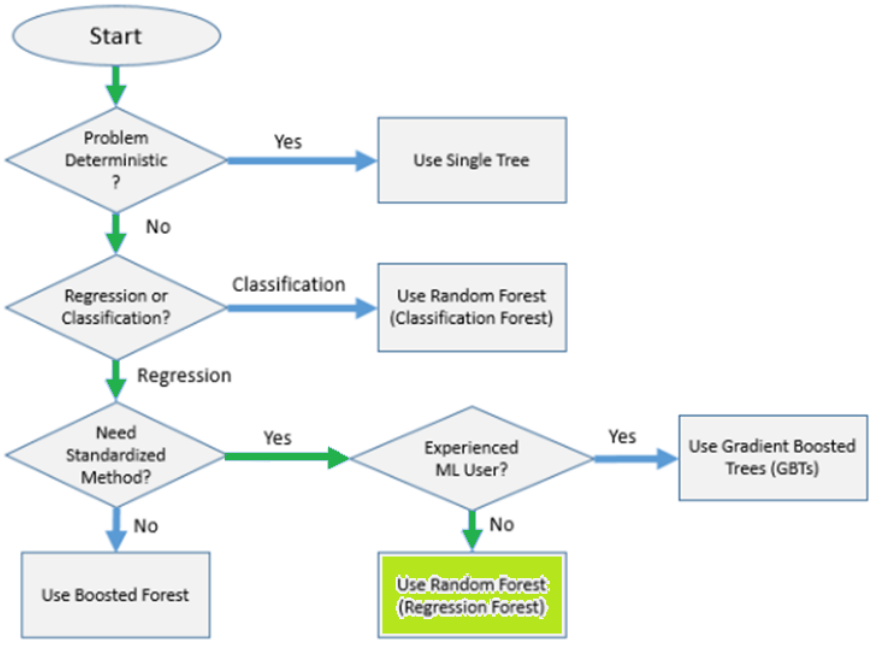


Figura 1.10: No caso em estudo "preço de imóveis" – Escolha: Random Forest - Regression Forest.

Isso funciona muito bem desde que o problema seja 'determinístico' - ou seja, as mesmas variáveis (features) sempre produzem os mesmos resultados e, há dados de treinamento suficientes, desencadeando no uso de uma Árvore Simples. Se o problema

for ‘estocástico’ - incorpora aleatoriedade nos dados ou resultados - uma Árvore de Decisão provavelmente não “generalizará” bem. Quanto ao conceito de “generalização” é importante entender a natureza da ‘População’ versus ‘Amostra’ (sample). Os dados de treinamento para um modelo de ML são quase sempre uma pequena amostra da população-alvo:

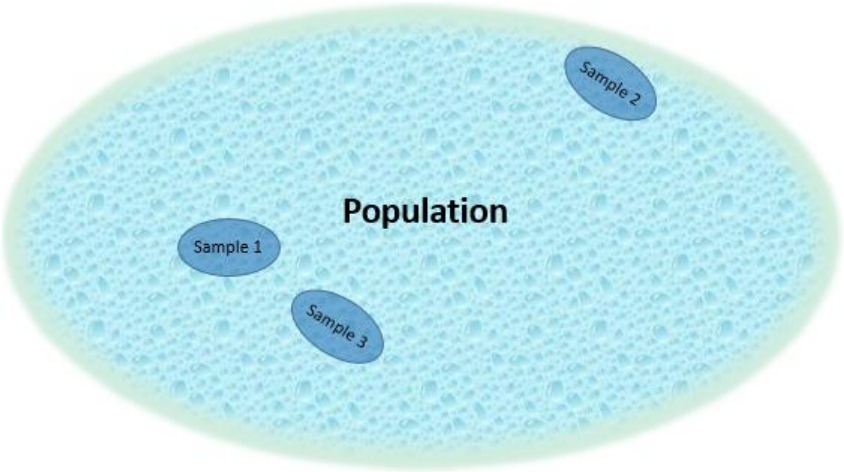


Figura 1.11: Generalização.

Em uma Random Forest, uma série de Árvores de Decisão são geradas, com alguma aleatoriedade adicionada ao processo de geração para garantir que cada árvore use um processo de decisão diferente na separação dos pontos. Então, para cada novo ponto de dados, todas as árvores são consultadas para sua previsão. Essas previsões individuais são agregadas para formar a previsão final.

Se o objetivo da Random Forest é escolher uma das várias classes (ou seja, uma ClassificationForest), então a agregação é feita por votação: a “classe prevista pelo maior número de árvores” torna-se a previsão final. Se o objetivo é prever um valor numérico (como o preço de imóveis), então temos uma RegressionForest, que forma uma previsão final pela “média das previsões das várias árvores”.

Um dos aspectos interessantes do Random Forest é que há muito poucos parâmetros para ajustar, e a escolha desses parâmetros geralmente tem muito pouca influência na precisão dos resultados. Bons resultados geralmente podem ser alcançados usando os parâmetros "default", tornando-o um dos algoritmos de ML mais fáceis de usar.

Outra grande característica é que o Random Forest pode lidar com muitas variáveis. Não é incomum usá-lo com milhares de variáveis.

Por fim, Random Forest são facilmente “paralelizáveis”, sendo, portanto, um algoritmo ideal para uso em clusters do HPCC Systems.

1.4.1. O Fluxo de Aprendizado de Máquina Supervisionado

No caso em estudo foi selecionado o bundle “LearningTrees ML” pelos seguintes motivos:

1. Fácil de usar:
   - Faz muito pouca suposição sobre a distribuição dos dados ou seus relacionamentos;
   - Ele pode lidar com muitos registros e muitos campos;

- Ele pode lidar com relações não lineares e descontínuas;
- Quase sempre funciona bem usando os parâmetros padrão e sem nenhum ajuste.

2. Escalabilidade
   - Escala bem em clusters HPCC Systems de quase qualquer tamanho.
3. Sua precisão de previsão é competitiva com os melhores algoritmos de última geração.

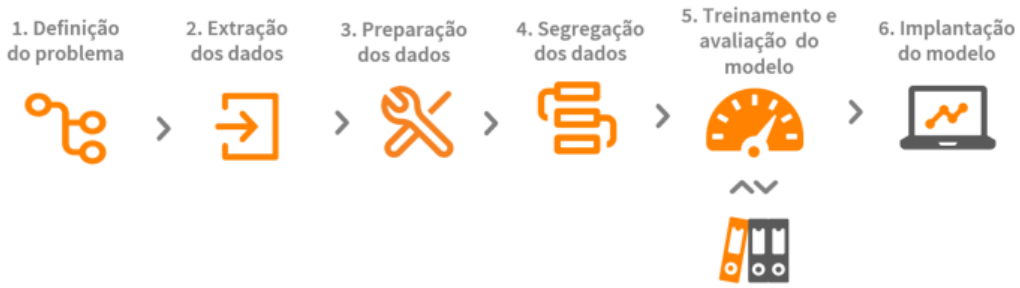


Figura 1.12: Fluxo de ML Supervisionado.

1.4.1.1. Definição do problema

"Dado um conjunto de atributos de uma propriedade (localização, metragem quadrada, ano de construção/aquisição, número de cômodos etc.), como predizer o seu valor real de venda?"

propertyid	house_numbe	house_nu	predir	street	streett	postdir	apt	city	state	zip	total_value	assessed_value	year_acquired	land_square_foot	living_square_fee	bedrooms	full_bath
828195	144			MCKIERNAN	DR			WALNUT CREEK	CA	94597	62614	62614	2006	20418	2485	3	2
1144455	281			CENTER	ST			BALTIMORE	MD	21136	105500	10550	2007	4807	1368	0	0
1494347	483			NEWTON	RD			FLAGSTAFF	AZ	86011	2220	2220	0	5654	1011	3	1
1910847	802			HATCHERY	CT			WOODLAND	WA	98674	356000	356000	0	6094	0	2	1
4267562	5007		E	ROY ROGERS	RD			TROY	MI	48085	327253	327253	2007	3484	0	3	0
4888602	7607			PEBBLESTONE	DR		000009	KERNVILLE	CA	93238	732179	732179	2010	19597	6132	6	6
48725	4			LONG	AVE			SUNRISE	FL	33323	271000	271000	2008	6880	2392	4	2
83528	6			TRILLUM	LN			WAYLAND	MA	02193	79889	79889	2007	7657	1657	4	1
94604	7			PARMENTER	AVE			PLYMOUTH	MN	55441	23800	23800	2005	19994	1754	3	2
220326	17			TIMBER	RD			LOS ANGELES	CA	90063	89000	89000	2008	7840	954	3	1
994609	212			FREYER	DR	NE		PHILOMONT	VA	20131	59800	59800	2009	11199	1241	3	0
1836173	724			EASTER	ST			ALLENTOWN	PA	18102	191600	191600	0	9100	2534	4	2
2910797	1903			SADDLE BROOK	DR			CLIO	CA	96106	61610	61610	2007	0	0	0	0
3083959	2158			RIVERSIDE	DR			UPPER MORELA…	PA	19006	90300	0	0	0	1235	3	2
3952189	4040			GRAND VIEW	BLVD		000054	RIO LINDA	CA	95673	0	0	0	2700720	0	0	0
4186238	4726			LAS PALMAS	CT			WAELDER	TX	78959	18816	18816	2009	2159	1320	0	0
4597143	6213			WILSON	RD			ZOLFO SPRINGS	FL	33890	72600	0	0	8496	0	3	1
4624905	6321			STONEWALL	LN			PATERSON	NJ	07514	139880	139880	2008	10454	1391	4	2
92326	7			KNOLLCREST	DR			NARANJA	FL	33032	76214	76214	2008	4800	930	2	0
1792852	704			ERIN	DR			TRABUCO	CA	92678	28010	28010	2007	5200	0	3	1
1843977	728		S	ARLINGTON HE…	RD			BLOOMING GRO…	TX	76626	130400	130400	2007	36154	1629	3	1
4214872	4821			MYRTLE OAK	DR		000025	SAN BERNARDI	CA	92376	32750		2007	93654	0	0	0

Figura 1.13: Preço do imóvel – Valor real de venda (total_value).

1.4.1.2. Extração dos dados

Será utilizado o dataset "Property.csv" no formato "CSV" contendo 1.662.959 registros. Usando os campos selecionados, tentaremos prever o preço da propriedade com base em outros dados (por exemplo: localização, metragem quadrada, ano de aquisição/construção, número de cômodos, código postal etc.).

Arquivo lógico: "~CLASS::XYZ::ML::Property".

- Códigos ECL utilizados:
  modProperty.ecl
  BWR_BrowseData.ecl

1.4.1.3. Preparação dos dados

A preparação dos dados é a etapa mais importante ao usar qualquer algoritmo ML. Como em qualquer processo de programação, a qualidade dos dados recebidos terá um grande efeito na qualidade dos resultados.

Aqui estão algumas regras importantes a serem consideradas:

- Os dados devem conter todos os valores numéricos;
- O primeiro campo no registro deve ser um identificador exclusivo (geralmente um ID de registro sem sinal - UNSIGNED);
- Para facilitar a implementação, mova seu campo "dependente" para o final da estrutura RECORD;
- Randomize seus dados para criar um recordset de treinamento e um teste mais preciso. Isso pode ser feito adicionando um campo com um número aleatório (RANDOM);
- Faça previamente a limpeza dos dados (cleasing);
- Usando o campo aleatório gerado, classifique e segregue os dados nos dados iniciais de Treinamento e de Teste. Os dados de Treinamento serão usados para treinar seu modelo de ML e os dados de Teste serão usados para avaliar (ou analisar) a eficácia do modelo. É fundamental que se reserve alguns dos dados para teste, pois é uma péssima ideia testar o modelo com os mesmos dados nos quais se treinou.

➢ Códigos ECL utilizados:
isCleanFilter.ecl
CleanProperty.ecl
modPrepData.ecl
BWR_ViewData1.ecl

##	propertyid	zip	assessed_value	year_acquired	land_square_footage	living_square_feet	bedrooms	full_baths	half_baths	year_built	total_value
1	79784	33424	76440	2015	4299	1255	3	2	0	2010	76440
2	3924129	20601	95900	2013	11224	1468	3	2	1	2007	95900
3	413843	8803	76000	2015	57000	1858	3	2	0	1970	76000
4	608224	98370	39340	2012	7405	1066	3	1	1	1967	39340
5	942963	72032	278400	2008	9600	2459	3	2	0	1963	278400
6	2237271	79935	143600	2011	8430	1008	2	1	1	1961	143600
7	4443742	84065	166934	2013	9317	1700	4	2	0	1991	166934
8	3834707	66227	348350	2012	15300	2663	4	2	1	2002	348350
9	3592739	19606	54000	2015	15060	2292	4	2	1	1980	90000
10	2916349	34639	119050	2015	6947	1709	3	2	0	2009	140950

Figura 1.14: Limpeza, padronização e consolidação de registros.

1.4.1.4. Segregação dos dados

Ao segregar os dados é importante coletar amostras aleatoriamente do seu dataset, ao invés de usar os primeiros 'N' registros no conjunto, porque pode existir alguma ordem oculta no processo que originalmente gerou os dados. A segregação dos dados de Treinamento e de Teste é feita facilmente usando ECL.

Como regra geral, cerca de 20 a 30% dos seus dados devem ser reservados para Teste. Entretanto, para fins didáticos e, principalmente, objetivando uma melhor performance durante a realização desse estudo, esse percentual será reduzido, considerando uma proporção de 5.000 registros para a amostra de Treinamento e 2.000 registros para a amostra de Teste:

```
// Considerando os primeiros 5.000 registros como amostra de Treinamento
myTrainData := PROJECT(PrepDataSort[1..5000], $.modPrepData.ML_Prop)
                    :PERSIST('~CLASS::XYZ::ML::Train');
// Considerando os 2.000 registros seguintes como amostra de Teste
myTestData  := PROJECT(PrepDataSort[5001..7000], $.modPrepData.ML_Prop)
                    :PERSIST('~CLASS::XYZ::ML::Test');
```

O próximo passo é converter os dados para o formato usado pelos pacotes configuráveis ML. Para operar genericamente com qualquer dado, o ML requer que os dados estejam em um "layout de matriz orientado a célula" conhecido como "NumericField". O bundle configurável ML_Core facilita essa tarefa:

```
IMPORT ML_Core;
// Conversão Matricial dos campos numéricos
ML_Core.ToField(myTrainData, myTrainDataNF);
ML_Core.ToField(myTestData, myTestDataNF);
```

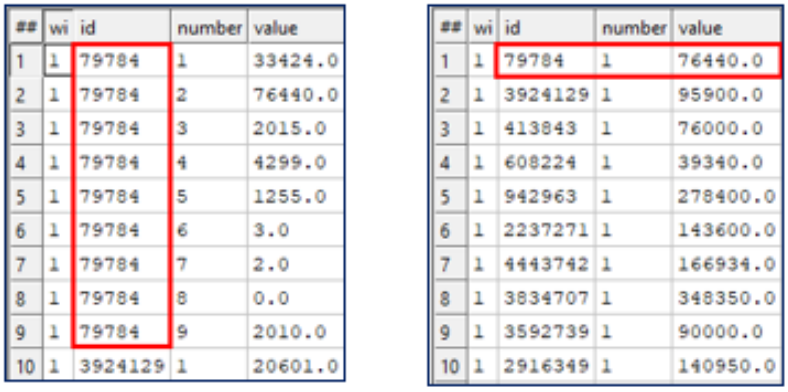
A etapa final antes de aplicar o modelo ML é separar os dados "independentes" dos dados "dependentes", definindo os dados do ML RECORD para colocar o campo de dados dependente no "final" do layout de registro. Um filtro simples nos dados de treinamento e de teste conclui esta etapa.

Sempre deve ser excluído o ID do registro exclusivo e, conte apenas seus campos independentes e dependentes. No caso, os dados de treinamento do dataset "Property", tem nove (9) campos independentes e o último campo (10º) é o campo dependente (o valor que estará sendo previsto).

O uso do PROJECT definindo o campo numérico = 1 não é estritamente necessário. Isso indica que é o primeiro campo dos dados dependentes. Como existe apenas um campo dependente, ele foi numerado de acordo:

```
EXPORT myIndTrainDataNF := myTrainDataNF(number < 10);
EXPORT myDepTrainDataNF := PROJECT(myTrainDataNF(number = 10),
TRANSFORM(RECORDOF(LEFT), SELF.number := 1, SELF := LEFT));
EXPORT myIndTestDataNF := myTestDataNF(number < 10);
EXPORT myDepTestDataNF := PROJECT(myTestDataNF(number = 10),
TRANSFORM(RECORDOF(LEFT), SELF.number := 1, SELF := LEFT));
```

- Códigos ECL utilizados:
  modSegConvData.ecl
  BWR_ViewData2.ecl



##	wi	id	number	value
1	1	79784	1	33424.0
2	1	79784	2	76440.0
3	1	79784	3	2015.0
4	1	79784	4	4299.0
5	1	79784	5	1255.0
6	1	79784	6	3.0
7	1	79784	7	2.0
8	1	79784	8	0.0
9	1	79784	9	2010.0
10	1	3924129	1	20601.0

##	wi	id	number	value
1	1	79784	1	76440.0
2	1	3924129	1	95900.0
3	1	413843	1	76000.0
4	1	608224	1	39340.0
5	1	942963	1	278400.0
6	1	2237271	1	143600.0
7	1	4443742	1	166934.0
8	1	3834707	1	348350.0
9	1	3592739	1	90000.0
10	1	2916349	1	140950.0

Figura 1.15: Nove (9) campos independentes e o último campo (10º) é o campo dependente.

1.4.1.5. Treinamento e avaliação do modelo

O primeiro passo é selecionar o modelo "learner". Para Regressão (quantitativa), será usado o módulo RegressionForest:

```
IMPORT $;
IMPORT LearningTrees AS LT;
/* Selecionando o algoritmo...
   Sintaxe:
RegressionForest(UNSIGNED numTrees=100, UNSIGNED featuresPerNode=0,
UNSIGNED maxDepth=100, SET OF UNSIGNED nominalFields=[]);
*/
myLearnerR := LT.RegressionForest();          // parâmetros default
```

Em seguida, o "learner" será usado para treinar e recuperar o modelo:

```
myModelR := myLearnerR.GetModel($.modSegConvData.myIndTrainDataNF,
    $.modSegConvData.myDepTrainDataNF);
```

Observe que o modelo obtido é uma estrutura de dados opaca (interpretável apenas pelo bundle) que encapsula todos os resultados do treinamento. O primeiro parâmetro fornecido para a função "GetModel" é o dataset de treinamento "independente" e o segundo parâmetro é o dataset de treinamento "dependente".

Em seguida, o modelo será usado para fazer previsões, prevendo o campo dependente com base nos campos de teste independentes:

```
predictedDeps := myLearnerR.Predict(myModelR,
$.modSegConvData.myIndTestDataNF);
```

Portanto, aplicando o modelo de treinamento e prevendo os resultados dependentes com base nos dados independentes, permitirá realmente produzir e visualizar esses resultados para análise.

Após o treinamento, será testado o modelo de previsão RegressionForest comparando-o com o dataset de teste dependente [myDepTestDataNF], que foi extraído anteriormente dos registros de teste segregados. ML_Core tem uma ótima maneira de fazer isso no módulo "Analysis". Esse módulo fornece uma avaliação genérica do poder preditivo do modelo. Também há uma grande variedade de outras métricas de avaliação fornecidas pelos pacotes individuais, específicas para os algoritmos desse pacote. No

Modelo de Regressão do LearningTrees, basta uma linha de ECL para realizar essa análise:

```
assessmentR := ML_Core.Analysis.Regression.Accuracy(predictedDeps,
$.modSegConvData.myDepTestDataNF);
```

A “Accuracy” usa o resultado da função Predict como primeiro parâmetro, comparando-o com os dados de teste dependentes que foi criado anteriormente. Aqui estão os resultados dos dados de treinamento para o dataset “Property” (vide Nota):

- Métricas chaves:
  - R2 (R-Quadrado - O "coeficiente de determinação". Aproximadamente, a proporção da variação nos dados originais que foram capturados pela regressão. O R-Quadrado pode variar ligeiramente de negativo a 1. Um R-Quadrado 1 significa que as previsões correspondem perfeitamente às reais. R-Quadrado zero ou abaixo significa que o modelo não tem valor preditivo. R-Quadrado 0,5 significa que metade da variação dos dados foi explicada pelo modelo.
  - MSE (Mean Squared Error - Erro quadrático médio) - O desvio médio quadrático entre os valores previstos e reais.
  - RMSE (Root Mean Squared Error - Raiz do Erro quadrático médio) - A raiz quadrada do MSE. Essa é uma expectativa do erro médio em uma determinada amostra.

##	wi	regressor	r2	mse	rmse
1	1	1	0.7304899830671003	7982069594.129144	89342.4288573416

Figura 1.16: Métricas chaves.

Nota: Para o conjunto de treinamento aplicado em “Property” (dataset bruto), a precisão foi um pouco abaixo de 30% na primeira tentativa, onde para melhorar a precisão foi necessário revisar os campos independentes. Após uma análise mais aprofundada, pode-se verificar que o campo [zip] não é realmente quantitativo, mas qualitativo (variável categórica). Então, a seguinte modificação no modelo de aprendizado foi realizada:

```
myLearnerR := LT.RegressionForest(,,,[1]);
```

A etapa final foi revisar as amostras e remover os registros com valores de campo “nulos” das amostras de treinamento e de teste usando o código [CleanProperty].

- Códigos ECL utilizados:

  BWR_TrainReg.ecl

1.4.1.6. Implantação do modelo

Essa etapa corresponde ao desenvolvimento de um serviço de consulta através da codificação de uma “função” (estrutura Function) da linguagem ECL, seguida da compilação do código no cluster Roxie:

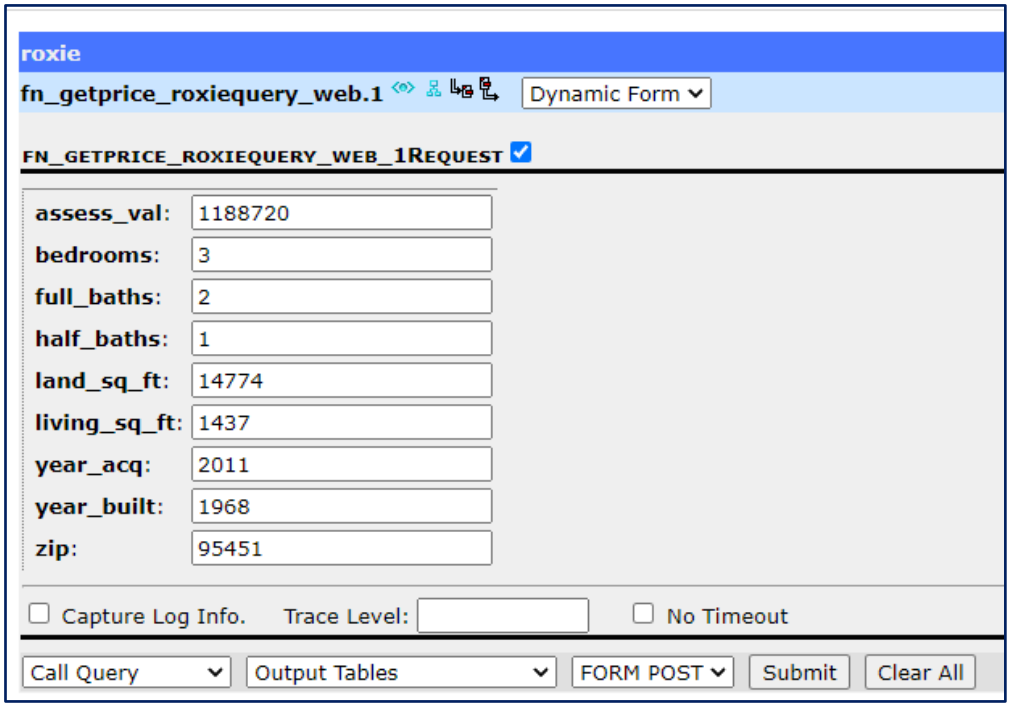


Figura 1.17: Carregamento de dados e disponibilização de serviço de consulta.

- Códigos ECL utilizados:

  fn_GetPriceReg.ecl
  BWR_TestQueriesReg.ecl

1.4.2. Conclusão

Algoritmos baseados em Árvores de Decisão são provavelmente os algoritmos mais poderosos e fáceis de usar no arsenal de ML. Se os dados forem numéricos, houver uma quantidade razoável de dados e o objetivo for "Previsão" ao invés de "Explicação", será difícil encontrar um melhor algoritmo do que Learning Trees.

Através desse estudo foi possível saber tudo o que se precisa saber para criar e testar modelos de ML com base nos dados disponíveis e usá-los para prever valores quantitativos (Regressão). Caso se deseje usar um bundle de ML diferente, se concluirá que todos os bundles operam de maneira muito semelhante, com algumas variações menores. Foi utilizado nesse estudo apenas os aspectos mais básicos dos bundles de ML.

A simplicidade conceitual de ML é um tanto enganadora. Cada algoritmo tem suas próprias peculiaridades (ou premissas e restrições) que precisam ser levadas em consideração para maximizar a precisão preditiva. Além disso, quantificar a eficácia das previsões requer habilidades em ciência de dados e estatística que a maioria não possui. Por esses motivos, é muito importante que não seja utilizada a exploração de ML para produzir produtos ou reivindicar habilidades sem antes consultar especialistas no campo.

1.4.3. Apêndice

- RoadMap
  - Procedimento - Sequência de criação e execução (submit) dos códigos:
    modProperty.ecl
    BWR_BrowseData.ecl (submit)
    isCleanFilter.ecl
    CleanProperty.ecl
    modPrepData.ecl
    BWR_ViewData1.ecl (submit)
    modSegConvData.ecl
    BWR_ViewData2.ecl (submit)
    BWR_TrainReg.ecl (submit)
    fn_GetPriceReg.ecl (compilar em Roxie)
    BWR_ViewData3.ecl (submit)
- Códigos ECL

[modProperty.ecl]

```
//
EXPORT modProperty := MODULE
  EXPORT Layout := RECORD
   UNSIGNED8 personid;
   INTEGER8  propertyid;
   STRING10  house_number;
   STRING10  house_number_suffix;
   STRING2   predir;
   STRING30  street;
   STRING5   streettype;
   STRING2   postdir;
   STRING6   apt;
   STRING40  city;
   STRING2   state;
   STRING5   zip;
   UNSIGNED4 total_value;
   UNSIGNED4 assessed_value;
   UNSIGNED2 year_acquired;
   UNSIGNED4 land_square_footage;
   UNSIGNED4 living_square_feet;
   UNSIGNED2 bedrooms;
   UNSIGNED2 full_baths;
   UNSIGNED2 half_baths;
   UNSIGNED2 year_built;
  END;
  EXPORT File := DATASET('~CLASS::XYZ::ML::Property',Layout,CSV);
END;
//
```

[BWR_BrowseData.ecl]

```
IMPORT $;
//
// Visualização da extração dos dados
OUTPUT($.modProperty.File);
COUNT($.modProperty.File);          // 1.662.959 registros
//
```

[isCleanFilter.ecl]

```
IMPORT $;
//
Property := $.modProperty.File;
//
// Limpando os dados...
EXPORT isCleanFilter := Property.zip <> '' AND
                        Property.assessed_value <> 0 AND
                        Property.year_acquired <> 0 AND
                        Property.land_square_footage <> 0 AND
                        Property.living_square_feet <> 0 AND
                        Property.bedrooms <> 0 AND
                        Property.full_baths <> 0 AND
                        Property.year_Built <> 0;
//
```

[CleanProperty.ecl]

```
IMPORT $;
//
Property := $.modProperty.File;
//
EXPORT CleanProperty := Property($.isCleanFilter);
//
// CleanProperty := Property($.isCleanFilter);
// OUTPUT(CleanProperty);
// COUNT(CleanProperty);          // 575.814 registros
//
```

[modPrepData.ecl]

```
IMPORT $;
//
Property := $.modProperty.File;
//
EXPORT modPrepData := MODULE
EXPORT ML_Prop := RECORD
UNSIGNED8 PropertyID;
UNSIGNED3 zip; // variável Categórica
UNSIGNED4 assessed_value;
UNSIGNED2 year_acquired;
UNSIGNED4 land_square_footage;
UNSIGNED4 living_square_feet;
UNSIGNED2 bedrooms;
UNSIGNED2 full_baths;
UNSIGNED2 half_baths;
UNSIGNED2 year_built;
UNSIGNED4 total_value; // variável Dependente (a ser determinada)
END;
//
  EXPORT ML_PropExt := RECORD(ML_Prop)
    UNSIGNED4 rnd;                // número randômico
  END;
//
EXPORT PrepData := PROJECT($.CleanProperty, TRANSFORM(ML_PropExt,
                                        SELF.rnd := RANDOM(),
                                        SELF.zip := (UNSIGNED3)LEFT.zip,
                                        SELF := LEFT))
                    :PERSIST('~CLASS::XYZ::ML::PrepData');
```

```
//
END;
//
```

[BWR_ViewData1.ecl]

```
IMPORT $;
//
// Preparação dos dados
OUTPUT($.modPrepData.PrepData);
COUNT($.modPrepData.PrepData);          // 575.814 registros
//
```

[modSegConvData.ecl]

```
IMPORT $;
IMPORT ML_Core;
//
PrepData := $.modPrepData.PrepData;
//
// Torne os dados aleatórios, ordenando os registros pelo número randômico
PrepDataSort := SORT(PrepData, rnd);
//
//
// Segregação dos dados - Considerando os primeiros 5.000 registros como amostra de Treinamento
myTrainData := PROJECT(PrepDataSort[1..5000], $.modPrepData.ML_Prop)
                    :PERSIST('~CLASS::XYZ::ML::Train');            // layout sem o campo rnd
//
// Segregação dos dados - Considerando os 2.000 registros seguintes como amostra de Teste
myTestData  := PROJECT(PrepDataSort[5001..7000], $.modPrepData.ML_Prop)
                    :PERSIST('~CLASS::XYZ::ML::Test');             // layout sem o campo rnd
//
//
// Conversão Matricial dos campos numéricos
ML_Core.ToField(myTrainData, myTrainDataNF);
ML_Core.ToField(myTestData, myTestDataNF);
//
//
EXPORT modSegConvData := MODULE
EXPORT myIndTrainDataNF := myTrainDataNF(number < 10);    // excluindo o campo propertyid
//
EXPORT myDepTrainDataNF := PROJECT(myTrainDataNF(number = 10),
TRANSFORM(RECORDOF(LEFT),
                                            SELF.number := 1,
                                            SELF := LEFT));
//
EXPORT myIndTestDataNF := myTestDataNF(number < 10);      // excluindo o campo propertyid
//
EXPORT myDepTestDataNF := PROJECT(myTestDataNF(number = 10),
TRANSFORM(RECORDOF(LEFT),
                                            SELF.number := 1,
                                            SELF := LEFT));
END;
//
```

[BWR_ViewData2.ecl]

```
IMPORT $;
//
// Segregação dos dados e Conversão matricial dos campos numéricos
```

```
OUTPUT($.modSegConvData.myIndTrainDataNF, NAMED('IndTrainData'));
OUTPUT($.modSegConvData.myDepTrainDataNF, NAMED('DepTrainData'));
OUTPUT($.modSegConvData.myIndTestDataNF,  NAMED('IndTestData'));
OUTPUT($.modSegConvData.myDepTestDataNF,  NAMED('DepTestData'));
//
```

[BWR_TrainReg.ecl]

```
IMPORT $;
IMPORT ML_Core;
IMPORT LearningTrees AS LT;
//
// Selecionando o algoritmo
//   Sintaxe: RegressionForest(UNSIGNED numTrees=100, UNSIGNED featuresPerNode=0,
//                             UNSIGNED maxDepth=100, SET OF UNSIGNED nominalFields=[])
// myLearnerR := LT.RegressionForest();          // parâmetros default
// myLearnerR := LT.RegressionForest(,,,[1]);    // zip não é realmente quantitativo, mas qualitativo
myLearnerR := LT.RegressionForest(10,,10,[1]);
//
//
// Obtendo o modelo treinado
myModelR :=
myLearnerR.GetModel($.modSegConvData.myIndTrainDataNF,$.modSegConvData.myDepTrainDataN
F);
OUTPUT(myModelR,,'~CLASS::XYZ::ML::myModelR', NAMED('Modelo_Treinado'),OVERWRITE);
//
//
// Testando o modelo
predictedDeps := myLearnerR.Predict(myModelR,$.modSegConvData.myIndTestDataNF);
OUTPUT(predictedDeps, NAMED('Valores_Previstos'));
//
//
// Avaliando o modelo
assessmentR :=
ML_Core.Analysis.Regression.Accuracy(predictedDeps,$.modSegConvData.myDepTestDataNF);
OUTPUT(assessmentR, NAMED('Avaliacao_do_Modelo'));
//
```

[fn_GetPriceReg.ecl]

```
IMPORT $;
IMPORT ML_Core;
IMPORT LearningTrees AS LT;
//
// Função de predição de preços de imóveis
EXPORT fn_GetPriceReg(Zip,
                      Assess_val,
                      Year_acq,
                      Land_sq_ft,
                      Living_sq_ft,
                      Bedrooms,
                      Full_baths,
                      Half_baths,
                      Year_built) := FUNCTION
//
// Transformação dos parâmetros de entrada no formato de ML data frame
  myInSet := [Zip, Assess_val, Year_acq, Land_sq_ft, Living_sq_ft, Bedrooms, Full_baths, Half_baths,
Year_built];
  myInDS := DATASET(myInSet, {REAL8 myInValue});
```

```
ML_Core.Types.NumericField PrepDataReg(RECORDOF(myInDS) Le, INTEGER cnt) :=
TRANSFORM
SELF.wi     := 1,
SELF.id     := 1,
SELF.number := cnt,
SELF.value  := Le.myInValue;
END;
myNewIndDataReg := PROJECT(myInDS, PrepDataReg(LEFT,COUNTER));
//
// Predição e retorno do valor do imóvel consultado
myModelR :=
DATASET('~CLASS::XYZ::ML::myModelR',ML_Core.Types.Layout_Model2,FLAT,PRELOAD);
myLearnerR := LT.RegressionForest(10,,10,[1]);
myPredictDeps := MyLearnerR.Predict(myModelR, myNewIndDataReg);
//
RETURN OUTPUT(myPredictDeps,{preco:=ROUND(value)});
END;
//
```

[BWR_TestQueriesReg.ecl]

```
IMPORT $;
//
// Teste da Função
// Zip | Assess_val | Year_acq | Land_sq_ft | Living_sq_ft | Bedrooms | Full_baths | Half_baths | Year_built
//
$.fn_GetPriceReg(95451,118720,2011,14774,1437,3,2,1,1968);
//
/*
assess_val:  118720
bedrooms:    3
full_baths:  2
half_baths:  1
land_sq_ft:  14774
living_sq_ft:1437
year_acq:    2011
year_built:  1968
zip:         95451
*/
//
```

1.5. Referências

Introduction to HPCC Systems Open Source Big Data Platform. Disponível em: https://cdn.hpccsystems.com/whitepapers/wp_introduction_HPCC.pdf Acesso em: 25 set.

Machine Learning Demystified. Disponível em: https://hpccsystems.com/resources/machine-learning-demystified/ Acesso em: 25 set. 2023.

HPCC Systems Machine Learning Library. Disponível em: https://hpccsystems.com/download/free-modules/hpcc-systems-machine-learning-library/ Acesso em: 25 set. 2023.

Using HPCC Systems Machine Learning. Disponível em: https://hpccsystems.com/resources/using-hpcc-systems-machine-learning/ Acesso em: 25 set. 2023.

ML_Core Documentation. Disponível em: https://cdn.hpccsystems.com/pdf/ml/ML_Core.pdf Acesso em: 25 set. 2023.

Source code: HPCC Systems ML_Core repository on GitHub. Disponível em: https://github.com/hpcc-systems/ML_Core Acesso em: 25 set. 2023.

Introduction to using PBblas on HPCC Systems. Disponível em: https://hpccsystems.com/resources/introduction-to-using-pbblas-on-hpcc-systems/ Acesso em: 25 set. 2023.

Documentation: PBblas Documentation. Disponível em: https://cdn.hpccsystems.com/pdf/ml/PBblas.pdf Acesso em: 25 set. 2023.

Source code: HPCC Systems PBblas repository on GitHub. Disponível em: https://github.com/hpcc-systems/PBblas Acesso em: 25 set. 2023.

Learning Trees — A guide to Decision Tree based Machine Learning. Disponível em: https://hpccsystems.com/resources/learning-trees-a-guide-to-decision-tree-based-machine-learning/ Acesso em: 25 set. 2023.

Documentation: LearningTrees Documentation. Disponível em: https://cdn.hpccsystems.com/pdf/ml/LearningTrees.pdf Acesso em: 25 set. 2023.

Source code: LearningTrees repository on GitHub. Disponível em: https://github.com/hpcc-systems/LearningTrees Acesso em: 25 set. 2023.