Aprendizado de Máquina para Previsão do Tempo de Execução de Aplicações Spark

Alexandre Maros[1], Jussara M. Almeida[1], Fabricio Murai[1]
Ana Paula Couto da Silva[1], Danilo Ardagna[2], Marco Lattuada[2]

[1]Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – Minas Gerais – Brasil

[2]Politecnico di Milano
Milão – Itália

{alexandremaros,jussara,murai,ana.coutosilva}@dcc.ufmg.br

{danilo.ardagna,marco.lattuada}@polimi.it

Abstract. *The rise of big data applications brought along a series of difficult challenges regarding the allocation of hardware and software resources. Typically these applications are known for being computationally expensive and having high heterogeneity on how they operate, making the task of estimating application's execution time very challenging. It may be still possible to correlate features extracted from the cloud environment and from the input dataset to the execution time. Such relationship may then be used to predict execution times. Based on such assumption, this work explores machine learning (ML) models to the task of predict execution time of Spark applications. This work investigates four ML models as well as different features, while also comparing their results against the current state-of-the-art. All models are evaluated in several scenarios and configurations, producing results that are significantly superior to the state-of-the-art in various cases.*

Resumo. *Aplicações de big data têm associada uma série de problemas desafiadores envolvendo a alocação de recursos de hardware e software. Tipicamente, essas aplicações são conhecidas por serem computacionalmente custosas e por possuírem uma grande heterogeneidade na forma em que operam, tornando desafiadora a tarefa de estimar os tempos de conclusão de resultados. É possível buscar uma relação entre atributos extraídos da configuração do ambiente de execução e do conjunto de dados de entrada e o tempo total de execução de certas aplicações. Tal relação pode ser utilizada na predição do tempo de execução. Este trabalho parte desta premissa e explora modelos de aprendizado de máquina para a previsão de tempos de execução de aplicações na plataforma Spark. O trabalho investiga quatro modelos e diferentes atributos, além de comparar os resultados com o estado-da-arte atual. Os modelos são avaliados em diversos cenários e configurações, produzindo resultados significativamente melhores que o estado-da-arte em alguns casos.*

1. Introdução

Nos últimos anos houve um aumento considerável no uso de algoritmos altamente distribuídos, que não só necessitam de um grande volume de dados como também são computacionalmente custosos. Esses algoritmos não só incluem as aplicações tradicionais

(como por exemplo consultas SQL), como também passam a contar com aplicações mais complexas, como processamento de linguagem natural [Stokes et al. 2015], análise de dados referentes à saúde pública [Zhang et al. 2017], entre outros.

Outra tecnologia que vem evoluindo juntamente com a ideia de processamento distribuído é a computação na nuvem. Houve na última década um aumento considerável na adoção dessa tecnologia. Usuários e empresas passaram a adotar servidores remotos para executar suas aplicações como uma alternativa viável ao antigo método de servidores locais, por ser uma alternativa, no geral, mais simples e barata [Low et al. 2011]. Em contrapartida, esses serviços que oferecem ambientes virtualizados para os mais variados usos (Amazon EC2, Azure), geralmente possuem uma extensa lista de possíveis configurações. O usuário deve escolher o número de instâncias (nós) que deseja, a quantidade de memória RAM, número de *cores*, entre outros. Tais escolhas afetam drasticamente o tempo de execução final e precisam ser cuidadosamente analisadas antes da aquisição do serviço. Para aplicações em que o tempo é essencial, como por exemplo em aplicações médicas, esse tipo de análise se torna ainda mais complexo e desafiador.

No contexto da investigação das capacidades de processamento massivo de dados, a avaliação de desempenho desses algoritmos têm uma importância vital para muitas áreas e há uma grande necessidade de se prever os tempos de execução dadas as mais diversas configurações. A habilidade de estabelecer prazos de entregas é essencial, especialmente no cenário de empresas. Com a previsão do tempo de execução, é possível estabelecer modelos, como otimizadores de recursos computacionais, para planejar configurações em tempo de execução, tendo um controle muito maior sobre o sistema [Zhang et al. 2011].

Uma das plataformas de programação distribuída mais utilizadas nos últimos anos é o Apache Spark [Zaharia et al. 2016]. Spark foi projetado como uma ferramenta de processamento analítico distribuído para um alto volume de dados. A ferramenta implementa as mais diversas funcionalidades, desde o processamento de dados relacionais, como consultas SQL [Armbrust et al. 2015], até algoritmos de aprendizado de máquina, como algoritmos de agrupamento e redes neurais [Meng et al. 2016].

Como o planejamento de capacidade e alocação de recursos são tarefas essenciais para sistemas distribuídos, particularmente big data e cloud computing, o problema a ser estudado é o de previsão de tempo de execução de aplicações Spark utilizando modelos de aprendizado de máquina supervisionado. O uso de aprendizado de máquina vem para capturar os comportamentos das aplicações de forma mais precisa que modelos tradicionais. Dada a entrada de n atributos (como tamanho do conjunto de dados, número de *cores*), espera-se que os modelos sejam capazes de prever com uma boa acurácia o tempo necessário para se executá-lo. Para validar os modelos, são coletados dois conjuntos de dados referentes à execução do algoritmo K-Means e de um algoritmo de aprendizado profundo em dois ambientes de execução distintos. Além disso, são considerados diversos cenários de interpolação e extrapolação do número de *cores* e do tamanho do conjunto de dados para verificar o comportamento dos modelos nesses casos. Para o K-Means e o algoritmo de aprendizado profundo, o MAPE (*Mean Average Percentage Error*) ficou, respectivamente, abaixo de 20% e 10% na maioria dos casos.

O restante deste artigo está organizado como segue. Trabalhos relacionados são discutidos na Seção 2. A Seção 3 descreve o cenário alvo estudado, definindo os ambi-

entes de execução e aplicações estudadas. Os modelos de previsão utilizados são apresentados na Seção 4. A Seção 5 mostra a metodologia de avaliação, com a descrição dos cenários, configurações e métrica. Os resultados e suas discussões são apresentados na Seção 6. Por fim, as conclusões e os trabalhos futuros são apresentados na Seção 7.

2. Trabalhos Relacionados

Diversos autores ao longo dos anos propuseram alternativas para a previsão e avaliação do tempo de execução e desempenho das plataformas de aplicações paralelas. Essas técnicas variam entre modelos analíticos, de simulação, híbridos ou até mesmo utilizando conceitos de aprendizado de máquina.

Quanto aos modelos analíticos, há diversas discussões na literatura que exploram modelos baseados em teoria de filas para capturar o comportamento das aplicações, podendo obter uma boa qualidade no serviço que está sendo utilizado [Menasce et al. 2004, Liang and Tripathi 2000, Mak and Lundstrom 1990]. Alguns modelos têm como premissa a execução sequencial dos *jobs* no sistema, outros estendem essa ideia e passam a considerar paralelismo entre dois ou mais *jobs*. Há também a proposta de modelos analíticos parametrizados a partir dos *logs* de execução do Spark [Pinto et al. 2018].

As previsões por simulação foram exploradas por outros autores. Song et al (2013) exploraram a previsão através de métodos de análise de *jobs* Hadoop. Uma outra abordagem é proposta por [Ardagna et al. 2016], onde a previsão é baseada em ferramentas de simulação já existentes: a JMT, baseada em teoria de filas e a GreatSPN, onde redes estocásticas baseadas em Petri Nets são utilizadas. Os modelos híbridos são mais recentes e trazem consigo técnicas de previsão que combinam tanto as técnicas de modelos analíticos, como também as técnicas de aprendizado de máquina [Popescu 2015].

Wang and Khan (2015) propuseram um modelo baseado em simulação para prever o tempo de execução de aplicações Spark. É realizada uma simulação do programa original com uma fração dos dados e os *logs* de execução são coletados. Dadas essas informações, são utilizados a sobrecarga de leitura e escrita dos dados, a memória total consumida e o tempo de execução da amostra para prever o desempenho de cada estágio. O modelo foi avaliado em um ambiente distribuído de 13 nós, obtendo-se uma boa acurácia em relação ao tempo de execução. O modelo foi testado em três algoritmos distintos: regressão logística, WordCount e K-Means.

Há poucos trabalhos na literatura que utilizam exclusivamente aprendizado de máquina na previsão de desempenho. Uma análise próxima a nossa pode ser encontrada em [Song et al. 2013], onde é proposto um método para prever o desempenho de uma aplicação *Map-Reduce* na plataforma Hadoop. Os autores utilizam uma série de modelos de regressão para prever a quantidade de tarefas que serão distribuídos pela aplicação. A ferramenta proposta é capaz de prever com uma boa acurácia a quantidade de trabalhos com pouca sobrecarga no sistema.

Por fim, um trabalho relacionado à previsão de tempo de execução de aplicações Spark e que é utilizado como base principal para este trabalho é atribuído a [Venkataraman et al. 2016]. Neste trabalho é proposto um modelo utilizando somente três atributos referentes ao tamanho do conjunto de dados e à quantidade de nós do ambiente de execução junto com o modelo de NNLS (*Non-Negative Least Squares*). Neste

trabalho, nós partimos dos resultados de [Venkataraman et al. 2016] e avaliamos os benefícios de se explorar alguns outros atributos obtidos a partir dos *logs* de execuções passadas, bem como outras técnicas de modelagem, buscando utilizar diferentes modelos de aprendizado de máquina para a previsão.

3. Cenário Alvo

Nesta seção são apresentados os principais conceitos relacionados ao cenário de previsão do tempo de execução de aplicações Spark estudado. A Seção 3.1 apresenta os principais componentes e o funcionamento da plataforma Spark para melhor entender os atributos utilizados como entrada para a previsão. A Seção 3.2 apresenta as aplicações alvo do estudo, enquanto a Seção 3.3 descreve os ambientes de execução utilizados.

3.1. Spark

O Apache Spark é uma plataforma de programação de propósito geral visando a execução de aplicações em ambientes distribuídos. O Spark se tornou uma das mais poderosas ferramentas para processamento de aplicações que utilizam grandes volumes de dados. Seu principal foco é fornecer aos usuários uma ferramenta de fácil acesso, rápida e que consiga englobar os mais diferentes tipos de aplicações de processamento de dados. Algoritmos de processamento de texto, aprendizado de máquina, consultas SQL, entre várias outras aplicações, são disponibilizados através de bibliotecas próprias [Karau et al. 2015].

A aplicação permite o processamento de dados em *batch*, onde os dados são previamente armazenados em um sistema de arquivo distribuído, como o HDFS (*Hadoop Distributed File System*). A menor unidade de execução da aplicação são as tarefas, que descrevem uma ação em uma partição de um RDD (*Resilient Distributed Datasets*), que é a abstração de armazenamento em memória do Spark, em que dados são organizados em blocos com tamanhos parecidos. Várias tarefas atribuídas para uma partição dos dados compõem um estágio. As tarefas são completamente independentes entre si.

O Spark implementa um modelo de execução baseado em um DAG (*Direct Acyclic Graph*), onde são modeladas as relações de precedência entre as tarefas que compõem a aplicação. Na execução, cada núcleo de processamento executa apenas uma tarefa por vez. Além disso, como as tarefas de um estágio são independentes, cada tarefa é alocada conforme a disponibilidade de recurso e a localização dos dados. Cada tarefa é alocada no momento em que um recurso do sistema se torna disponível [Karau et al. 2015].

3.2. Aplicações

Duas aplicações foram consideradas para o estudo dos tempos de execução. A primeira aplicação é o algoritmo de agrupamento K-Means, que é componente central de inúmeras aplicações de aprendizado de máquina por viabilizar e direcionar análises preliminares dos conjuntos de dados. A segunda aplicação, SparkDL, é baseada em métodos de classificação muito utilizados e que são considerados como estado-da-arte para muitas aplicações, sendo elas as redes neurais convolucionais profundas e o classificador SVM.

K-Means [Arthur and Vassilvitskii 2007] é um algoritmo de aprendizado não supervisionado que tem por objetivo agrupar os dados de entradas em k grupos. O algoritmo determina o posicionamento de k centroides em um espaço de várias dimensões com o

objetivo de minimizar a distância entre esses centroides e os dados disponíveis. Este algoritmo possui uma alta complexidade computacional devido à sua aleatoriedade e ao grande número de cálculos de distância entre pontos, tornando assim, um interessante candidato à análise do seu comportamento. A implementação utilizada é retirada do Spark-Bench[1], uma coleção de algoritmos para testar o desempenho do Spark.

A segunda aplicação é baseada na biblioteca de *deep learning* SparkDL[2]. Tal biblioteca suporta algoritmos de redes neurais profundas, assim como abordagens mais populares, como aprendizado por transferência, onde um modelo previamente treinado é utilizado. A tarefa realizada pela aplicação é a de classificação binária de imagens, onde inicialmente é utilizado a rede convolucional InceptionV3 previamente treinada para a extração dos atributos e em sequência é feito o treinamento de um modelo SVM linear [Hsu et al. 2008] como a camada de classificação.

Os algoritmos K-Means e SparkDL possuem processos complexos e uma grande quantidade de estágios e tarefas. Além disso, o número de iterações realizadas depende da convergência para os dados de entrada. K-Means foi executado gerando um conjunto de dados sintético com 100 atributos e variando o número de pontos entre 5, 10, 15 e 20 milhões. Ao analisar o DAG do K-Means, foi verificado que o mesmo possui 15 estágios para todos os tamanhos do conjunto de dados considerados. Já o algoritmo SparkDL referente à classificação de imagens foi executado com 1000, 1500 e 2500 imagens.

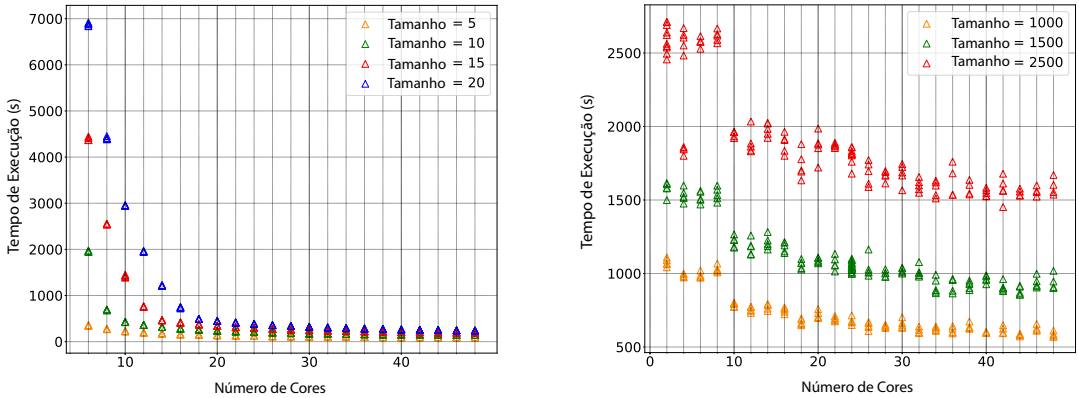
3.3. Ambientes de Execução

Os dados utilizados em nosso estudo foram extraídos de *logs* de execução das aplicações alvo em dois ambientes distintos: Microsoft Azure e IBM Power8. A ideia foi explorar dois ambientes com características diferentes. O primeiro ambiente, Microsoft Azure, é um servidor de nuvem pública, que é afetado pelo compartilhamento de recursos e é caracterizado por sua variabilidade. O segundo servidor, IBM Power8, é um servidor privado da Politecnico di Milano, que foi dedicado exclusivamente para executar as aplicações sem nenhuma outra atividade em execução. Na plataforma da Microsoft Azure, as execuções das aplicações foram feitas utilizando a configuração D13v2, que oferece 8 *cores* por CPU e 56GB de memória e Spark 2.2.0 num ambiente Linux. A plataforma IBM Power8 dispõem de 4 máquinas virtuais, com 12 *cores* e 58GB de RAM.

Dada as aplicações e os ambientes de execução, cada aplicação foi executada nos dois ambientes e os *logs* de execução foram salvos. Para cada configuração do ambiente, cada aplicação foi executada dez vezes. Esses *logs* foram então pré processados por um *parser* para extração das informações referentes aos atributos que serão utilizados (descritos abaixo) e o tempo de execução total. Os tempos de execução de cada aplicação em função da quantidade de *cores* e do tamanho do conjunto de dados são mostrados nas Figura 1(a) e 1(b) respectivamente, onde cada ponto mostra o tempo total de uma única execução da aplicação. Nota-se a melhoria no desempenho com o aumento do número de cores, como esperado. Mais ainda, nota-se uma variabilidade muito pequena entre os resultados de múltiplas execuções do K-Means para a mesma configuração (a maioria dos pontos estão sobrepostos), enquanto que as execuções do SparkDL exibem maior variabilidade no tempo de execução.

[1] `https://codait.github.io/spark-bench`

[2] `https://databricks.github.io/spark-deep-learning/site/index.html`



(a) K-Means (conjunto de dados em milhões de pontos)

(b) SparkDL (conjunto de dados em número de imagens classificadas)

Figura 1. Tempo de Execução em segundos para cada algoritmo de acordo com a quantidade de *cores* e tamanho do conjunto de dados

4. Modelos de Previsão

Nesta seção são apresentados os modelos de previsão propostos assim como o modelo de referência, chamado Ernest, proposto em [Venkataraman et al. 2016]. A tarefa de previsão explorada neste artigo é um tarefa de regressão, na qual se deseja inferir uma função (linear ou não) que tenha como entrada um conjunto de atributos e produzir como saída uma variável resposta (tempo de execução). A Seção 4.1 apresenta os atributos utilizados pelos algoritmos de previsão enquanto a Seção 4.2 apresenta os algoritmos de regressão utilizados.

4.1. Atributos

A Tabela 1 apresenta os atributos explorados por cada modelo analisado, inclusive pelo modelo de referência Ernest. O Ernest utiliza apenas 4 atributos de entrada que são derivados do número de cores e do tamanho do conjunto de dados. Para as novas soluções de previsão do tempo de execução foram propostos dois conjuntos de atributos: Reduzido e Completo, também descritos na Tabela 1. O conjunto reduzido de atributos (CRA) representam informações *a priori* da execução da aplicação e são baseadas no modelo Ernest. O tamanho do conjunto de dados e o número de *cores* são as informações básicas sobre a aplicação, o *log* do número de cores é referente ao padrão de agregação dos estágios que segue uma estrutura de árvore, e por fim, a razão entre o tamanho do conjunto de dados e o número de *cores* captura o tempo de processamento paralelo dos algoritmos.

O Conjunto Completo de atributos (CCA) é a junção do conjunto CRA com métricas adicionais referente ao DAG da execução da aplicação. Esses atributos são o número de tarefas necessárias para completar a aplicação, o mínimo, máximo, média e desvio padrão do número de tarefas de cada estágio, a média e o máximo de tempo de execução das tarefas e a média e o máximo do tempo de *shuffle*, i.e., tempo relacionado à distribuição dos dados entre os vários núcleos de processamento. Estas informações estão disponíveis apenas *a posteriori*, i.e., *após* a execução do algoritmo. Portanto, estes

Conjunto de Atributos	Atributos
Ernest [Venkataraman et al. 2016]	razão entre o tamanho do conjunto de dados e o número de *cores* log do número de cores raiz quadrada da razão entre o tamanho do conjunto de dados e o número de *cores* a razão do tamanho do conjunto de dados ao quadrado e o número de cores
Conjunto Reduzido de Atributos	tamanho do conjunto de dados número de *cores* log do número de cores razão entre o tamanho do conjunto de dados e o número de *cores*
Conjunto Completo de Atributos	todos os atributos do conjunto reduzido numero de tarefas min/max/média/desvio padrão do número de tarefas média/max tempo de execução das tarefas média/max tempo de *shuffle*

Tabela 1. Atributos utilizadas nos modelos

atributos adicionais aqui considerados correspondem a *estimativas* destas medidas. Especificamente, dados uma aplicação alvo e um conjunto de logs de execuções prévias da mesma, o modelo de regressão *Random Forest* [Liaw et al. 2002] é utilizado para estimar os valores dos atributos adicionais com base nestes dados históricos e no conjunto reduzido de atributos. As estimativas são então utilizadas como entrada (i.e., atributos adicionais) para a tarefa de previsão do tempo de execução. Por um lado, a inclusão destes atributos adicionais pode trazer mais informação útil e melhorar a eficácia da previsão do tempo de resposta; por outro, erros nas suas estimativas podem contribuir para adicionar ruído na entrada do previsor e, consequentemente, levar a erros de previsão maiores.

4.2. Algoritmos de Previsão

Para os novos métodos previsão de tempo de execução, foram considerados quatro modelos de regressão para a previsão do tempo de execução, sendo eles: Linear Regression, Decision Tree, Redes Neurais e Random Forest [Robert 2014]. Linear Regression foi escolhido pela fácil interpretabilidade, Decision Tree e Random forest também pela interpretabilidade e pela captura de comportamentos não lineares. Já as Redes Neurais têm a vantagem de conseguirem capturar interações não-triviais entre os atributos. Explorar diferentes modelos é essencial para verificar a diferença de comportamento de cada aplicação para entender qual modelo melhor se adéqua a previsão em cada cenário.

O modelo de referência Ernest [Venkataraman et al. 2016] utiliza o Linear Regression como algoritmo de regressão, e os coeficientes da função de previsão são calculados a partir do NNLS (*Non Negative Least Squares*). O NNLS é uma variação do algoritmo de *Least Squares* que possui a restrição de que os coeficientes não podem ser negativos. A vantagem é a de que os tempos previstos nunca podem ser negativos. Entretanto, como pode ser visto na avaliação experimental, um único modelo não consegue capturar o comportamento de diferentes aplicações, tendo a necessidade de explorar outros algoritmos.

5. Metodologia de Avaliação

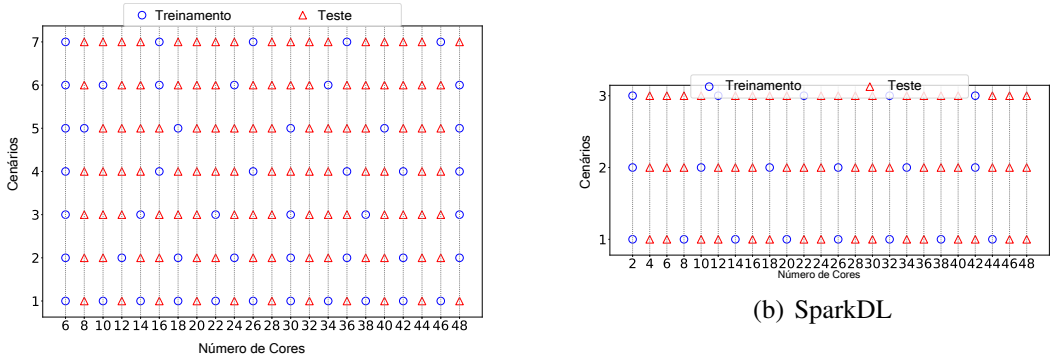
Os quatro algoritmos de regressão aqui considerados são técnicas de aprendizado de máquina supervisionadas. Logo o desenvolvimento de cada modelo requer a divisão dos dados em um conjunto de treino e um conjunto de teste. O conjunto de treino é usado para aprender o modelo de previsão (função e hiperparâmetros), e deve conter dados da entrada (atributos) e da saída (tempo de execução) de várias execuções passadas. Já o conjunto

de teste é utilizado para avaliar o modelo. Os atributos são utilizados como entrada do modelo, e o tempo previsto é comparado com o valor real.

Foram projetados diferentes cenários de divisão dos dados entre treino e teste, conforme descrito na Seção 5.1. Além disto, diferentes tamanhos de conjunto de dados também foram considerados, como apresentado na Seção 5.2. Em todos os casos, os hiperparâmetros dos modelos foram escolhidos com base em um *grid-search* (i.e., busca exaustiva) utilizando o algoritmo validação cruzada 3-Fold sobre os dados de treinamento. Os valores de hiperparâmetros que levaram aos melhores resultados no treino foram utilizados na fase de teste, quando a eficácia dos modelos foi avaliada, utilizando a métrica descrita na Seção 5.3.

5.1. Cenários de Avaliação

Foram definidos vários cenários para avaliar a capacidade de interpolação de cada modelo utilizando diferentes configurações de *cores* no conjunto de treino e no conjunto de teste. Os cenários definidos para cada aplicação estão representados na Figura 2 e foram elaborados de forma que casos de índice maior fossem associados a previsões mais difíceis, já que seus dados de treinamento incluem amostras de um menor intervalo de número de cores. Os círculos indicam dados de execuções usadas no treinamento, enquanto os triângulos indicam aqueles usados no teste. Para o K-Means há sete cenários de teste e para o SparkDL há três cenários de teste. O primeiro cenário é aquele que contém a maior variedade de número de *cores* no conjunto de treinamento. Isto é, considerando o K-Means, dados para 6, 10, 14, 18, 22, 26, 30, 34, 38, 42 e 46 cores são usados para treinamento (e validação cruzada), enquanto dados para os demais números de cores são usados no conjunto de teste. Para os casos subsequentes os *cores* são retirados gradativamente do treino. Isso permite analisar como as previsões são afetadas para os casos de interpolação do número *cores*.



(a) K-Means

(b) SparkDL

Figura 2. Cenários de avaliação de acordo com o número de *cores* para as duas aplicações

5.2. Configurações

Além do teste de interpolação de *cores*, também foram definidos testes de interpolação e extrapolação em relação ao tamanho do conjunto de dados. As Tabelas 2 e 3 mostram,

Número da Configuração	Tamanho do conjunto de dados no Treino	Tamanho do conjunto de dados no Teste
1	5	5
2	20	20
3	5, 10, 15	20
4	5, 10, 20	15
5	5, 15, 20	10
6	10, 15, 20	5

Tabela 2. Tamanho do conjunto de dados (em milhões de pontos) no treino e teste para K-means

Número da Configuração	Tamanho do conjunto de dados no Treino	Tamanho do conjunto de dados no Teste
1	1000	1000
2	1500	1500
3	2500	2500
4	1000 e 1500	2500
5	1000 e 2500	1500
6	1500 e 2500	1000

Tabela 3. Tamanho do conjunto de dados (número de imagens) no treino e teste para SparkDL

respectivamente, as configurações de tamanhos de conjunto de dados para os algoritmos K-Means e SparkDL.

Para o algoritmo K-Means, as configurações 1 e 2 são referentes ao tamanho do conjunto de dados fixo, a configuração 3 é para o teste de extrapolação do tamanho do conjunto de dados e as restantes para a interpolação. No SparkDL têm-se as configurações 1, 2 e 3 com um conjunto de dados fixo, 4 e 6 para extrapolação e 5 para interpolação.

5.3. Métrica de Avaliação

A métrica de comparação é o MAPE (*Mean Average Percentage Error*) definido por:

$$MAPE = \frac{100}{N} \sum_{k=1}^{N} \left| \frac{y_k - \hat{y}_k}{y_k} \right| \tag{1}$$

onde N é o número de pontos no conjunto de teste, y_k é o valor real do tempo de execução e $\hat{y}_k$ o tempo previsto pelo modelo. Para todos os resultados apresentados, a métrica de desempenho MAPE é calculada utilizando cinco execuções de cada modelo.

6. Resultados

Esta seção discute os resultados da previsão do tempo de execução. As avaliações exploram o uso dos modelos utilizando dois conjuntos distintos de atributos (CRA e CCA), nos diferentes cenários descritos na Seção 5.1. Além disso também são investigadas a extrapolação e a interpolação do tamanho do conjunto de dados utilizando diferentes configurações treino-teste desses conjuntos, descritos na Seção 5.2. Por fim, comparamos os resultados obtidos com o estado da arte, o modelo Ernest [Venkataraman et al. 2016].

6.1. K-Means

Nesta seção analisamos os resultados obtidos para a aplicação Spark MLlib K-Means. São consideradas seis configurações de tamanhos de conjunto de dados, apresentadas

na Tabela 2, assim como sete cenários de ambientes de execução distintos que diferem pela quantidade de *cores* disponíveis para a execução, conforme ilustrado na Figura 2(a). Considerando as seis configurações e sete cenários, tem-se 42 combinações.

As Tabelas 4 e 5 apresentam a métrica MAPE para a configuração 2, com a maior quantidade de dados (20 milhões de pontos), considerando o CCA (conjunto completo de atributos) e CRA (conjunto reduzido de atributos), respectivamente. Na Tabela 4, o modelo Linear Regression resulta em MAPE de até uma ordem de magnitude maior do que os outros modelos. Decision Tree e Random Forest tiveram desempenho semelhante, apresentando os melhores resultados dentre todos os modelos, com exceção do cenário 7, para o qual a Rede Neural obteve o menor MAPE. Analisamos a importância média atribuída a cada atributo na classificação do Random Forest: 28% ao número de *cores*, 9% ao tempo máximo de execução de uma tarefa, 24% à média do tempo de execução da tarefa, 24% à média do tempo de *shuffle* e 15% ao tempo máximo de *shuffle*. Como 72% da importância vem de atributos que devem ser estimados por outro modelo isso pode implicar na piora da previsão devido a introdução de ruído por parte deste segundo modelo. Quando há diferentes tamanhos de conjuntos de dados no treino, cada um dos 14 atributos totais assume uma importância semelhante, de aproximadamente 7%.

	Decision Tree	**Linear Regression**	**Neural Network**	**Random Forest**
Cenário 1	19.13	141.54	47.22	20.44
Cenário 2	16.98	86.42	28.27	17.28
Cenário 3	31.76	155.62	54.80	30.62
Cenário 4	29.27	190.65	33.29	35.09
Cenário 5	23.74	169.78	30.50	23.86
Cenário 6	21.17	169.56	33.51	22.28
Cenário 7	28.12	228.61	21.95	28.70

Tabela 4. MAPE (%) do tempo de execução estimado no Power8 para K-means e Configuração 2 (20M pontos no treino e 20M pontos no teste) usando o conjunto completo de atributos

Entretanto, o MAPE para a previsão do tempo de execução com o CRA é drasticamente menor, como pode ser visto na Tabela 5, tendo reduzido de 228% para 32.20% no Cenário 7 (para o modelo Linear Regression). Os resultados para os modelos Decision Tree e Random Forest são similares. Os únicos casos em que o desempenho foi pior com o CRA foram aqueles com as Redes Neurais nos Cenários 5 e 7. No caso do Random Forest, a importância atribuída aos atributos na predição é: 30% ao número de *cores*, 37% ao log do número de máquinas e 33% à razão entre o tamanho do conjunto e o número de *cores*. Em configurações onde há diferença de tamanho entre conjuntos, a importância do número de *cores* caí em 10%, que passam a integrar a importância atribuída ao tamanho.

É interessante notar que os modelos com o conjunto reduzido de atributos superam os modelos construídos a partir do conjunto completo. Isto indica que os atributos derivados do DAG, enquanto estimativas previstas de execuções passadas, podem ser muito ruidosos e degradar significativamente a previsão do tempo de execução.

Dado o desempenho superior do modelo Random Forest (o modelo também obteve os melhores resultados nas outras configurações, mas a análise detalhada foi omitida por restrições de espaço), nós focamos nesta solução, utilizando o conjunto reduzido de atributos, para comparação com o estado-da-arte Ernest. Os resultados das duas estratégias, para todos os cenários e configurações, são mostrados na Tabela 6. Inici-

	Decision Tree	Linear Regression	Neural Network	Random Forest
Cenário 1	13.19	35.10	42.70	6.69
Cenário 2	16.49	30.60	11.72	14.57
Cenário 3	23.73	32.20	26.96	17.12
Cenário 4	26.31	31.68	32.72	20.92
Cenário 5	21.13	32.13	56.33	18.55
Cenário 6	17.38	36.20	22.08	15.11
Cenário 7	23.43	32.20	30.60	21.12

Tabela 5. MAPE (%) do tempo de execução estimado no Power8 para K-means e Configuração 2 (20M pontos no treino e 20M pontos no teste) usando o conjunto reduzido de atributos

almente, é possível notar que, em todos os casos, o modelo proposto obteve resultados consideravelmente melhores que o Ernest, especialmente nas Configurações 2 e 6. O modelo proposto foi capaz de produzir bons resultados mesmo para os casos onde há pouca quantidade de dados para o treinamento (últimos cenários) tanto nos casos de extrapolação como de interpolação do conjunto de dados. Com exceção da configuração 6, o MAPE obtido pelo Random Forest ficou próximo ou menor do que 20%.

	Config. 1		Config. 2		Config. 3		Config. 4		Config. 5		Config. 6	
	RF	Ernest	RF	Ernest	RF	Ernest	RF	Ernest	RF	Ernest	RF	Ernest
Cenário 1	3.09	6.48	6.69	405.04	11.61	78.81	13.59	44.37	17.50	80.02	33.06	220.35
Cenário 2	3.60	6.10	14.57	402.35	13.63	81.83	10.63	56.59	20.56	82.06	33.58	181.93
Cenário 3	4.73	14.48	17.12	465.79	12.30	109.26	15.34	63.56	16.69	102.03	31.56	203.86
Cenário 4	4.49	8.62	20.92	441.36	12.55	104.38	15.37	65.67	14.52	90.92	65.34	247.54
Cenário 5	5.27	26.89	18.55	585.99	14.46	168.76	13.20	69.91	12.33	143.69	32.27	323.82
Cenário 6	3.00	29.50	15.11	586.85	11.65	142.55	11.00	84.57	20.04	118.48	34.95	248.86
Cenário 7	4.05	19.11	21.12	510.56	12.45	125.77	14.89	63.39	22.46	125.83	73.48	242.15

Tabela 6. K-Means – Resultados do Random Forest com conjunto reduzido de atributos e modelo Ernest para todos os cenários e configurações considerados

De modo geral, para a aplicação K-Means, o modelo Linear Regression obteve as piores previsões. Esse resultado não é surpreendente dado que o modelo depende da premissa de linearidade dos dados. Como visto na Figura 1(a), o tempo não aumenta linearmente com o tamanho do conjunto de dados. Os três outros modelos que não assumem linearidade obtiveram melhores resultados.

6.2. SparkDL

Os resultados da previsão do tempo de execução da aplicação de aprendizado profundo e processamento de imagens SparkDL é discutido nesta seção. No total são analisadas seis configurações de tamanho do conjunto de dados, apresentadas na Tabela 3, e três cenários de ambientes de execução nos conjuntos de treino e testes, apresentados na Figura 2(b), totalizando 18 combinações.

A Tabela 7 mostra a métrica MAPE da previsão do tempo de execução para a Configuração 4 e o conjunto completo de atributos. A Rede Neural foi o modelo que obteve os melhores resultados, tendo um MAPE menor que 30% em todos os casos. Os outros três modelos exibiram um resultado semelhante entre si, com erro aproximadamente 40% superior ao da rede neural. A importância dos atributos para o modelo de árvore se mostrou semelhante ao caso anterior, tanto para o CCA quanto para o CRA.

	Decision Tree	Linear Regression	Neural Network	Random Forest
Cenário 1	36.55	37.96	26.68	37.52
Cenário 2	42.80	43.29	27.24	41.74
Cenário 3	40.52	42.38	24.92	41.24

Tabela 7. MAPE (%) do tempo de execução estimado no Microsoft Azure para SparkDL e Configuração 4 (1000 e 1500 imagens no treino e 2500 no teste) usando o conjunto completo de atributos

A Tabela 8 apresenta os resultados considerando o CRA onde há uma melhora considerável para o modelo Linear Regression. Sem os atributos adicionais, o modelo conseguiu um desempenho extremamente bom, conseguindo obter um MAPE ainda menor que o das Redes Neurais com o CCA. A Random Forest e a Decision Tree obtiveram resultados semelhantes aos anteriores, com o CCA. Esses resultados corroboram a evidência anterior de que os atributos do DAG acabam introduzindo ruído nas previsões.

	Decision Tree	Linear Regression	Neural Network	Random Forest
Cenário 1	36.92	7.86	26.80	37.18
Cenário 2	42.48	7.38	18.27	41.68
Cenário 3	40.79	7.78	10.65	40.59

Tabela 8. MAPE (%) do tempo de execução no Microsoft Azure para SparkDL e Configuração 4 (1000 e 1500 imagens no treino e 2500 no teste) usando o conjunto reduzido de atributos

A Tabela 9 compara o Ernest com o modelo que obteve o melhor resultado, i.e., Linear Regression com o conjunto reduzido de atributos. Para a aplicação SparkDL não houve tanta diferença entre o modelo proposto e o Ernest como houve com a aplicação K-Means. A métrica MAPE se manteve muito semelhante na maior parte dos casos, com grandes diferenças apenas no Cenário 2 e Configuração 4 e nos Cenários 2 e 3 para a Configuração 6, onde o modelo Linear Regression obteve os melhores resultados.

	Config. 1		Config. 2		Config. 3		Config. 4		Config. 5		Config. 6	
	LR	Ernest	LR	Ernest	LR	Ernest	LR	Ernest	LR	Ernest	LR	Ernest
Cenário 1	7.19	6.78	5.18	6.26	5.32	6.94	7.86	8.66	5.67	6.65	17.91	17.06
Cenário 2	5.89	6.39	6.23	6.83	7.37	7.69	7.38	14.15	8.45	8.06	5.26	12.00
Cenário 3	5.46	6.86	5.36	7.19	6.43	7.17	7.78	8.66	7.76	6.97	5.53	14.45

Tabela 9. SparkDL – Resultados do Linear Regression com conjunto reduzido de atributos e modelo Ernest para todos os cenários e configurações considerados

Na aplicação SparkDL o modelo Linear Regression obteve os melhores resultados com o CRA, o que sugere uma maior linearidade dos dados. A Rede Neural obteve o segundo melhor resultado, com erro particularmente próximo ao do melhor modelo no Cenário 3 na Configuração 4. Os modelos Decision Tree e Random Forest não conseguiram capturar tão bem a relação entre os dados e o tempo de execução na aplicação e consequentemente não obtiveram bons resultados quando comparados com os outros modelos. Por outro lado, o Ernest obteve resultados semelhantes aos obtidos com os atributos aqui propostos e não teve uma diferença tão grande na aplicação K-Means, sugerindo que a escolha de apenas um algoritmo de estimação de parâmetros (NNLS no caso do Ernest) não é suficiente para obter a melhor previsão em todos os casos.

7. Conclusões e Trabalhos Futuros

Neste artigo avaliamos a acurácia da previsão do tempo de execução de aplicações Spark com base em modelos de aprendizado de máquina. Consideramos duas aplicações de *big data* que são executadas em um ambiente distribuído na plataforma Spark e propusemos dois conjuntos de atributos: um conjunto reduzido de atributos (CRA), e um conjunto completo (CCA), que estende o CRA com atributos que são estimativas de propriedades estruturais do DAG. Além disso, foram construídos cenários para verificar a capacidade de interpolação e extrapolação da quantidade de *cores* e dos tamanhos dos conjuntos de dados. Por fim, também comparamos tais com o atual estado de arte para a previsão de desempenho dessas aplicações, o modelo Ernest [Venkataraman et al. 2016].

Os resultados se mostraram promissores, superando o desempenho do estado da arte em alguns casos e obtendo resultados muito semelhantes em outros. Para a aplicação K-Means, alguns dos modelos combinados com o CRA obtiveram resultados significativamente melhores, reforçando a ideia de que há a necessidade de modelos mais complexos que não assumam a linearidade dos dados sendo analisados, como é o caso do Ernest. A investigação dos atributos relacionados ao DAG na composição do CCA mostraram que as mesmas acabaram introduzindo uma quantidade de ruído, piorando as previsões na maioria dos casos. O conjunto de atributos reduzidos, que levam apenas as informações *a priori* da execução geraram resultados melhores.

Como trabalhos futuros, pretendemos explorar mais aplicações, como o SVM e PCA, para verificar o comportamento de cada modelo aqui proposto. Além disso é de grande interesse avaliar o comportamento dos algoritmos em ambientes mais diversos, verificando se as propriedades dos mesmos são alteradas quando executados em configurações distintas. Por fim, resta estudar um possível otimizador de configurações de ambientes na nuvem com base na saída dos modelos.

Referências

Ardagna, D., Bernardi, S., Gianniti, E., Aliabadi, S. K., Perez-Palacin, D., and Requeno, J. I. (2016). Modeling performance of hadoop applications: A journey from queueing networks to stochastic well formed nets. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 599–613. Springer.

Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., et al. (2015). Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM.

Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Hsu, C., Chang, C., and Lin, C. (2008). A practical guide to support vector classification. *BJU International*, 101:1396–1400.

Karau, H., Konwinski, A., Wendell, P., and Zaharia, M. (2015). *Learning spark: lightning-fast big data analysis*. "O'Reilly Media, Inc.".

Liang, D.-R. and Tripathi, S. K. (2000). On performance prediction of parallel computations with precedent constraints. *IEEE Transactions on Parallel and Distributed Systems*, 11(5):491–508.

Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

Low, C., Chen, Y., and Wu, M. (2011). Understanding the determinants of cloud computing adoption. *Industrial management & data systems*, 111(7):1006–1023.

Mak, V. W. and Lundstrom, S. F. (1990). Predicting performance of parallel computations. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):257–270.

Menasce, D. A., Almeida, V. A., Dowdy, L. W., and Dowdy, L. (2004). *Performance by design: computer capacity planning by example*. Prentice Hall Professional.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241.

Pinto, T. B., da Silva, A. P. C., and Almeida, J. M. (2018). Previs ão do tempo de resposta de aplicações de big data em ambientes de nuvem. In *Simpósio Brasileiro de Redes de Computadores (SBRC)*, volume 36.

Popescu, A. D. (2015). Runtime prediction for scale-out data analytics. Technical report, EPFL.

Robert, C. (2014). Machine learning, a probabilistic perspective.

Song, G., Meng, Z., Huet, F., Magoules, F., Yu, L., and Lin, X. (2013). A hadoop mapreduce performance prediction method. In *2013 IEEE International Conference on High Performance Computing and Communications*, pages 820–825. IEEE.

Stokes, C., Kumar, A., Choi, F., and Weischedel, R. (2015). Scaling nlp algorithms to meet high demand. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 2839–2839. IEEE.

Venkataraman, S., Yang, Z., Franklin, M. J., Recht, B., and Stoica, I. (2016). Ernest: Efficient performance prediction for large-scale advanced analytics. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 363–378.

Wang, K. and Khan, M. M. H. (2015). Performance prediction for apache spark platform. In *2015 IEEE 17th International Conference on High Performance Computing and Communications*, pages 166–173.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., et al. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

Zhang, Q., Zhu, Q., and Boutaba, R. (2011). Dynamic resource allocation for spot markets in cloud computing environments. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 178–185.

Zhang, Y., Qiu, M., Tsai, C.-W., Hassan, M. M., and Alamri, A. (2017). Health-cps: Healthcare cyber-physical system assisted by cloud and big data. *IEEE Systems Journal*, 11(1):88–95.