UNIVERSIDADE PRESBITERIANA MACKENZIE

PAULO SIMPLÍCIO BRAGA

PROCESSAMENTO DE LINGUAGEM NATURAL APLICADO À ANÁLISE DE LOGS DE SISTEMA

São Paulo - SP
2020

PAULO SIMPLÍCIO BRAGA

PROCESSAMENTO DE LINGUAGEM NATURAL APLICADO À ANÁLISE DE LOGS DE SISTEMA

Trabalho apresentado ao curso de Big Data e Visualização de Dados, área de Tecnologia da Informação da UNIVERSIDADE PRESBITERIANA MACKENZIE, como requisito parcial para a Obtenção do grau de especialista em Aprendizagem de Máquina em Inteligência Artificial.

São Paulo - SP
2020

RESUMO

Mitigar o impacto de uma falha computacional se torna possível quando predições de falhas acuradas são empregadas. Toda uma engenharia preventiva pode agir quando sabe-se que o sistema está no limiar de um evento de falha e assim, medidas podem ser tomadas afim de não haver grande impacto para o usuário. Pensando em situações onde existe um grande número de dispositivos rodando com diversos componentes, um sistema preditivo de falhas pode mostrar-se bastante útil. O grande desafio é montar e modelar um sistema capaz de fazer previsões com um bom tempo de antecedência.

A discussão aqui proposta visa analisar a viabilidade de um modelo para predição de falhas, baseado em análise de mensagens de log de sistema. Tais mensagens apresentam mudanças no sistema que podem indicar que algum componente não está se comportando da maneira esperada. Mensagens isoladas de log podem não dizer nada, contudo uma sequência ou um certo padrão delas pode ser suficiente para prever alguma falha no sistema.

Keywords: Linux, syslog, NLP

SUMÁRIO

1 **INTRODUÇÃO**

Um sistema operacional como Kernel/Linux gera uma grande carga de mensagens de log a cada segundo com intuito de reportar a saúde do sistema como um todo e de cada um dos dispositivos que o compõe. Por mais rico que seja esse diagnóstico, decifrá-lo não é das coisas mais triviais, nem mesmo para os usuários mais experientes.

Devido a sua natureza *open source,* o Kernel/Linux torna-se muito atrativo quando se tem como proposta embarcar um sistema operacional de tempo real em dispositivos diversos, tal qual um decodificador de sinal para TV digital que será utilizado para estudo de caso na presente dissertação.

A discussão que quero trazer com a proposta aqui apresentada é possibilidade de relacionar tudo o que é reportado pelo sistema e criar um modelo capaz de prever, com certo grau de eficiência, quando um módulo poderá vir a apresentar algum tipo de falha. O objeto de estudo dessa dissertação é o disco rígido externo contudo, com o que aqui é proposto, não há dúvidas quanto a possibilidade de generalizar a aplicação para diferentes camadas do sistema.

2 **DESENVOLVIMENTO**

Um sistema Linux embarcado, possui característica específicas. Porém, tanto uma distribuição comum encontrada em servidores e desktops, quanto uma embarcada em uma placa com um fim específico, fornecem uma grande quantidade de mensagens que registram o estado do sistema e de seus componentes. Estas são as chamadas mensagens de log

O problema aqui estudado é relacionado a falhas no disco rígido externo em um decodificador de sinal digital e como pode-se aproveitar as mensagens geradas nos momentos que antecedem tais falhas.

Utilizando técnicas para vetorizar palavras e posteriormente aplicar modelos de aprendizagem de máquina, é possível encontrar padrões e anomalias dentro do sistema. Nos capítulos à seguir serão apresentados alguns conceitos técnicos para se alcançar o que é proposto na presente dissertação, assim como uma sugestão para a solução do problema

2.1 O KERNEL/LINUX

Quando se fala de Linux no dia a dia, normalmente a referência é a uma distribuição Linux. Tais distribuições variam de propósito e tamanho, contudo possuem um objetivo comum: Fornecer ao usuário um algo pronto, com sistema de arquivos definido e um processo de instalação, tanto para o seu sistema operacional (Kernel) quanto para todos os outros softwares construídos em cima dele, que permita seu funcionamento em um determinado hardware (YAGHMOUR et al., 2008, p. 25).

2.1.1 **Linux embarcado**

Linux embarcado é comumente entendido como um sistema completo, ainda que o termo "embarcado" não signifique um formato especial de Kernel/Linux. Na verdade, a mesma versão do sistema operacional kernel pode estar presente em uma variedade de dispositivos diferentes, de servidores com uma grande capacidade de processamento a um simples dispositivo de monitoramento qualquer com memória limitada. O que diferencia os sistemas são suas configurações, que estarão de acordo com o fim proposto.

Um Sistema embarcado pode ser descrito, em síntese, como uma placa contendo todos os componentes que montam um computador - da maneira que o público no geral está acostumado - como processador, memórias e dispositivos de

entrada e saída. Seu funcionamento está condicionado as limitações de seu hardware, em contraste com a modularidade e propósito generalista de desktops e notebooks.

O Kernel/Linux embarcado é configurado tal qual as limitações do sistema no qual ele é inserido, bem como tais configurações têm como objetivo atender a necessidade operacional proposta. Por exemplo, o Kernel/Linux pode estar embarcado em um dispositivo de monitoramento de temperatura, que basicamente terá um sensor e uma interface serial para comunicação, ou pode-se pensar em um sistema de áudio automotivo com interfaces bluetooth e w-fi para comunicação e uma interface para processamento de áudio e vídeo. Enfim, a aplicação para um sistema Linux embarcado é muito versátil e sua única limitação é o preço que o cliente final quer pagar por ele.

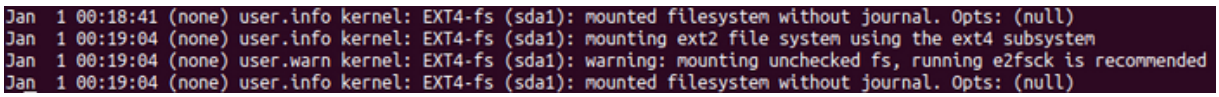
2.1.2 **Os logs do sistema**

Muitas das aplicações registram dados sobre suas ações e status. Estes registros, chamados de log, são guardados no sistema de arquivos do Linux, normalmente em '/var/syslog/', mas o usuário pode escolher um outro local que faça mais sentido para ele.

Para escrever no arquivo de log, uma aplicação chamada syslogd é utilizada. Ela irá ler a mensagem de log, que deve estar formatada de acordo com um padrão definido, e escrever no arquivo de log.

O syslog é o formato onipresente no sistemas Linux, podendo ser encontrado em todas as suas variantes (MATOTEK; TURNBULL; LIEVERDINK, 2017).

Figura 1 - Mensagens de log caputradas pelo syslogd



Fonte: O autor (2020)

Como pode ser visto na figura 1, cada linha de entrada no arquivo de log, é constituída pela data, usuário, nível da mensagem e o conteúdo, propriamente dito.

As mensagens podem ser caracterizadas por níveis que variam desde informações apenas, até mensagens de emergência, quando o sistema esta no limiar de um problema mais sério. A figura 2 ilustra tais níveis.

Figura 2 - Níveis de log do Kernel

Name	String	Meaning	alias function
KERN_EMERG	"0"	Emergency messages, system is about to crash or is unstable	pr_emerg
KERN_ALERT	"1"	Something bad happened and action must be taken immediately	pr_alert
KERN_CRIT	"2"	A critical condition occurred like a serious hardware/software failure	pr_crit
KERN_ERR	"3"	An error condition, often used by drivers to indicate difficulties with the hardware	pr_err
KERN_WARNING	"4"	A warning, meaning nothing serious by itself but might indicate problems	pr_warning
KERN_NOTICE	"5"	Nothing serious, but notably nevertheless. Often used to report security events.	pr_notice
KERN_INFO	"6"	Informational message e.g. startup information at driver initialization	pr_info
KERN_DEBUG	"7"	Debug messages	pr_debug, pr_devel if DEBUG is defined
KERN_DEFAULT	"d"	The default kernel loglevel	
KERN_CONT	""	"continued" line of log printout (only done after a line that had no enclosing \n) [1]	pr_cont

Fonte: Debugging... (2019)

2.2 PROCESSAMENTO DE LINGUAGEM NATURAL

O Processamento Natural de Linguagem, diz respeito a uma série de técnicas que envolvem a aplicação de métodos estatísticos, com ou sem *insights da* linguística, para entender textos com o propósito de resolver problemas do mundo real. Este "entendimento" deriva principalmente da transformação de texto em representações computacionais, que são estruturas combinacionais discretas ou contínuas tais quais vetores ou tensores, gráficos, e árvores (RAO; MCMAHAN, 2019).

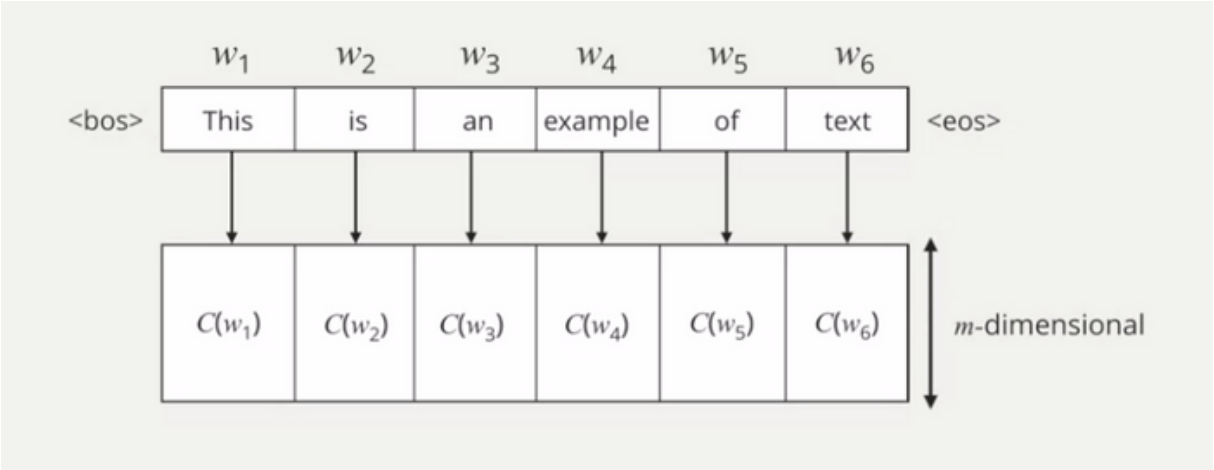
2.2.1 **Word Embedding - A Vetorização das Palavras**

Dentro do processamento de linguagem natural, um dos conceitos chave é o de vetorização das palavras. Na prática, isto significa que todas as palavras do vocabulário são mapeadas para vetores, para que a análise possa ser feita.

A ideia de modelar ou mapear palavras para vetores é comumente chamada

de **Word2Vec (**palavra para vetor, em tradução livre). Este processo de mapeamento também é conhecido como *embedding* (embarcar, em português) e tal nomenclatura é largamente utilizada no universo de NLP (Natural Language Processing). Uma vez feito o mapeamento das palavras em *m* dimensões (Figura 3), têm-se a representação matemática das palavras e é possível inferir a relação entre elas , levando em consideração a distância no espaço vetorial.

Figura 3 - Representação do método Word2Vec



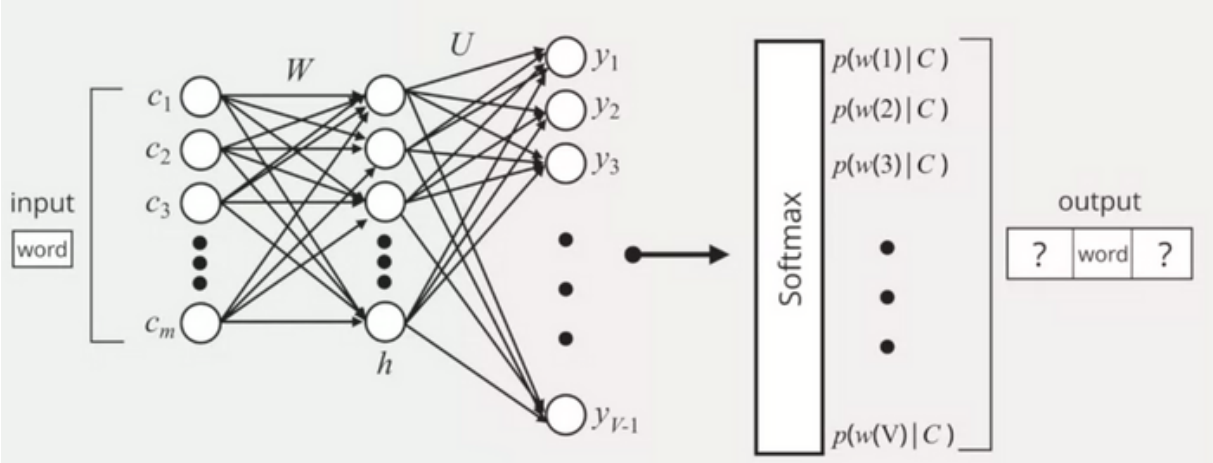
Fonte: Carin (2019)

Existem diversos métodos e tecnologias para aplicação do conceito aqui apresentado para relacionar palavras com vetores e, geralmente, os resultados são bastante similares. Isto demonstra, de maneira implícita, que tal conceito é bastante robusto.

2.2.2 **Modelos Neurais para Reconhecimento de Texto**

A ideia por trás de fazer a vetorização, é ter uma maneira para saber o que está ao redor de cada uma das palavras em um dado documento, ou seja, saber a relação entre as palavras. E uma vez vetorizada, a é possível aplicar um modelo neural de múltiplas camadas como o *Multi Layer Perceptron.*

Figura 4 - Modelo Neural para Texto



Fonte: Carin (2019)

O Multi Layer Perceptron aqui utilizado, como exemplificado na Figura 4, é um modelo neural com duas camadas intermediárias, uma entrada e uma saída. A entrada é o vetor de uma palavra qualquer do vocabulário que se está trabalhando, e a saída apresenta quais palavras estão na vizinhança do que foi passado na entrada.

Os valores de h na primeira camada do Perceptron são definidos de acordo com o demonstrado na equação 1, onde **W** é o valor do peso, **b** é o viés (ambos definidos pelo algorítimo de treinamento), **c** é o vetor e **tanh** é a função de ativação que irá atuar em cada um dos componentes.

Equação 1 - equação para encontrar h

$$h = tanh([\sum W.c] + b) \tag{1}$$

Fonte: O autor (2020)

A função de ativação tomada como referência é a tangente hiperbólica (tanh), descrita na equação 2, contudo existem diferentes funções de ativação que podem ser utilizadas de acordo com a melhor acurácia para o modelo em questão.

Equação 2 - Tangente Hiperbólica

$$tanhx = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2}$$

Fonte: O autor (2020)

A saída da camada representada por **h** é então multiplicada por um outro

peso **U** para se chegar, conforme demonstrado na Figura 4, aos valores de y, que vão de y(1) até y(v-1), onde v é o tamanho do vocabulário. A equação 3 demonstra este processo.

Equação 3 - Encontrando os valores de y

$$y_v = (\sum U_v * h_d) + b \tag{3}$$

Fonte: O autor (2020)

Por fim, na saída do modelo, os valores encontrados na equação 3 passam pela função de ativação Softmax, que irá dizer quais são as palavras próximas, no espaço vetorial, da palavra de entrada.

A função Softmax tem como resultado valores entre 0 e 1. E ela também divide cada saída pela soma de todas as saídas, dando uma distribuição probabilística discreta como resultado (RAO; MCMAHAN, 2019, p. 48). Levando em consideração o exemplo da Figura 4, para um vocabulário de tamanho **v**, a representação da função ficará conforme demonstrado na equação 4.

Equação 4 - Função softmax

$$softmax(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{v} e^{y_j}} \tag{4}$$

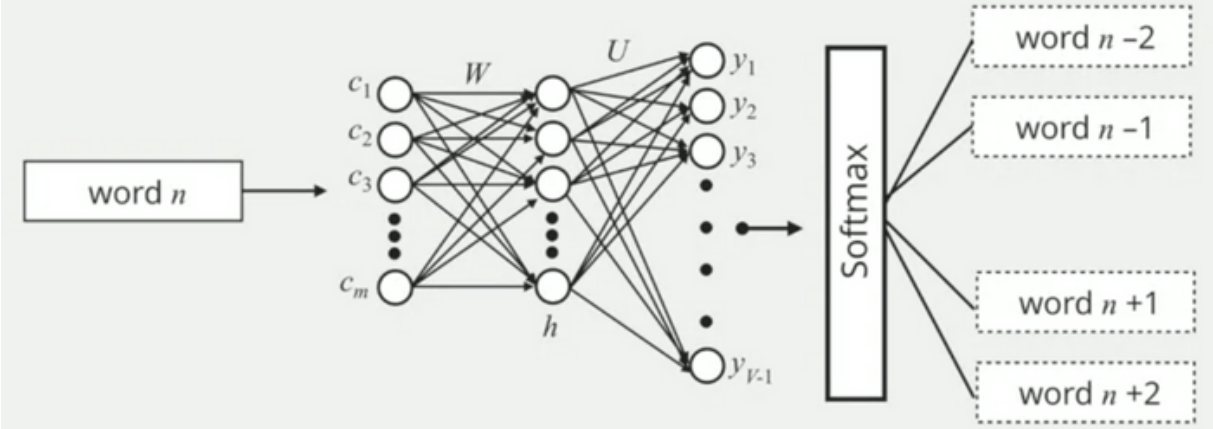
Fonte: O autor (2020)

2.2.3 **CBOW e Skip-Gram**

Os dois tipos de algoritmos largamente utilizados em processamento de linguagem natural são o CBOW e o Skip-Gram.

Dado um conjunto de sentenças, o modelo fará a iteração em cada uma das palavras de cada uma das sentenças e utilizará a palavra de entrada do modelo para prever as palavras vizinhas, no caso do Skip-Gram, conforme demonstrado na figura 5.

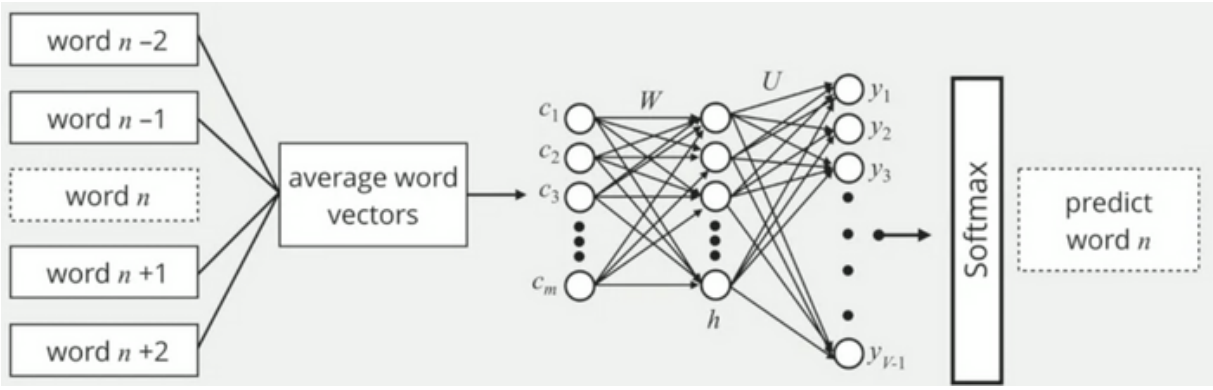
Figura 5 - Modelo Skip-Gram



Fonte: Carin (2019)

Já o CBOW - cujo o acrônimo significa Continuous Bag Of Words (Saco De Palavras Contínuo, em tradução livre) - trabalho o conjunto de sentenças de modo que, dado um contexto de palavras de tamanho pré-definido, a saída será a palavra a ser prevista. A Figura 6 demonstra como é o modelo CBOW.

Figura 6 - Modelo CBOW



Fonte: Carin (2019)

2.3 PROPOSTA DE IMPLEMENTAÇÃO

Visto a capacidade que algorítimos e modelos de processamento de linguagem natural possuem, faz sentido para o autor da presente dissertação, buscar propor uma abordagem para fazer uso deste poder computacional em tarefas técnicas e árduas tal qual monitorar de forma preventiva a saúde de um sistema com Kernel/Linux embarcada.

Na sequência do texto, serão explorados métodos teóricos para a implementação de um sistema neural de processamento de linguagem natural para,

juntamente com outras análises textuais, prever quando um disco rígido externo de um decodificador de sinal digital, operando com um sistema linux embarcado, irá apresentar defeito de operação. Para atingir o objetivo proposto, a ideia é trabalhar em cima dos arquivos de log de sistema (syslog).

2.3.1 **Exemplo de espaço vetorial do syslog**

O primeiro passo a ser dado pode ser, conforme demonstrado nos capítulos anteriores, a conversão das palavras para um espaço vetorial. Como a mensagem salva pelo syslog segue alguns parâmetros do sistema Kernel/Linux, o processamento dela possui algumas singularidades.

No capítulo 2.1.2 foi demonstrado como é dividida cada linha salva no arquivo gerado pelo syslog (Tabela 1). A ideia é aproveitar cada segmento da mensagem de maneira diferente e estabelecer uma relação entre eles.

Tabela 1 - Segmentação das mensagens de log

Data	Usuário	Nível	Mensagem
Jan 1 00:06:13	root	user.info	mounting ext2 file system using the ext4 subsystem
Jan 1 00:07:20	root	user.warn	warning: mounting unchecked fs

Fonte: O autor (2020)

Utilizando como referência a Tabela 1, pode-se fazer a vetorização do conteúdo da coluna 'Mensagem' e, à partir dai, criar relações entre todas as palavras no corpo do texto. Assim, é possível dizer quais palavras estão ligadas aos diferentes momentos de funcionamento do disco rígido externo.

Para ilustrar o processo de convergência para o espaço vetorial, utilizo aqui uma parte do corpo do log de sistema (Figura 7).

Figura 7 - Trecho do log para vetorização

```
Jan  1 00:06:13 (none) user.crit kernel: GISB: timeout at 0xa2a7021 [W timeout], core: rdc_0
Jan  1 00:06:13 (none) user.info kernel: EXT4-fs (sda1): mounting ext2 file system using the ext4 subsystem
Jan  1 00:06:13 (none) user.warn kernel: EXT4-fs (sda1): warning: mounting unchecked fs, running e2fsck is recommended
Jan  1 00:06:13 (none) user.info kernel: EXT4-fs (sda1): mounted filesystem without journal. Opts: (null)
Jan  1 00:07:20 (none) user.warn kernel: EXT4-fs (sda1): warning: mounting unchecked fs, running e2fsck is recommended
Jan  1 00:07:20 (none) user.info kernel: EXT4-fs (sda1): mounted filesystem without journal. Opts: (null)
```

Fonte: O autor (2020)

Com o auxílio da biblioteca para vetorização de palavras gensim, utilizando um código escrito em Python3, é possível obter a relação numérica compreendida em um intervalo de 0 até 1, sendo que quanto mais próximo de 1, maior a relação entre as palavras. Utilizando como referência a palavra 'ext4', a Figura 8 demonstra

o resultado deste processo.

Figura 8 - Exemplo de palavras próximas a 'ext4'

```
1 model.wv.most_similar('ext4')[:3]
```

```
[('recommended', 0.9780221581459045),
 ('null', 0.9105072021484375),
 ('e2fsck', 0.8021650314331055)]
```

Fonte: O autor 920200

Para facilitar a visualização, foi utilizado um espaço vetorial de 3 dimensões. Mais uma vez utilizando 'ext4' como exemplo, é possível observar os valores de seu vetor na Figura 9.

Figura 9 - Valores Vetoriais de 'ext4'

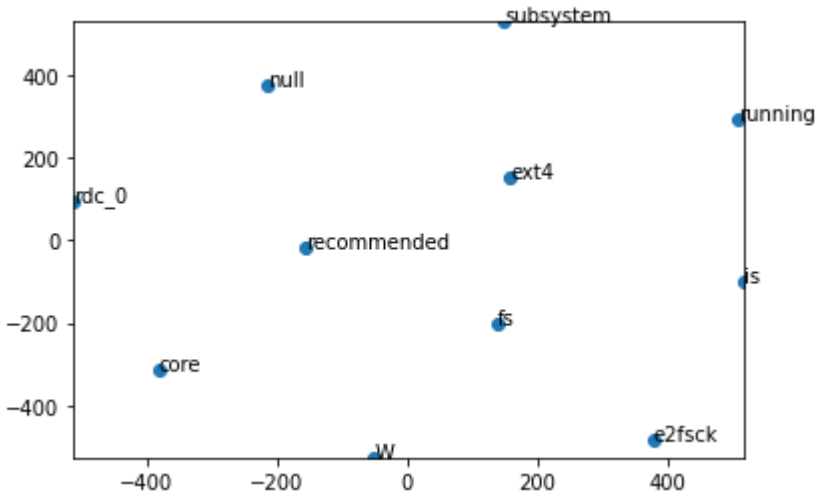
```
1 model.wv['ext4']
```

```
array([-0.12689888, -0.11589331, -0.07932106], dtype=float32)
```

Fonte: O autor (2020)

Por fim, uma maneira bastante didática para enxergar as proximidades das palavras no corpo do texto, é a plotagem gráfica em um plano cartesiano. Para tal, prossigo utilizando como referência a palavra 'ext4', conforme demonstrado na figura 10.

Figura 10 - Visualização Gráfica de 'ext4'



Fonte: O autor (2020)

Com essas informações em mãos, já é possível começar a ter um

entendimento do que está acontecendo com o sistema, mais especificamente, no caso aqui debatido, com o disco rígido externo.

2.3.2 **Trabalhando com os demais componentes**

Uma vez que tem-se os vetores da coluna mensagem, conforme demonstrado anteriormente, surge então a pergunta: Como é possível utilizar essa informação para resolver o problema de identificação de falhas no disco rígido externo? A verdade é que sozinhos, os vetores de proximidade não devem conseguir resolver este problema. É preciso criar uma relação com outros aspectos e a ideia aqui proposta, é que tal relação seja criada com a data/horário e o nível de criticidade da mensagem , utilizando a Tabela 1 como referência. Nota-se que a mesma tabela também traz a coluna usuário, contudo não há motivo para utilizá-la, pois não existe ali nenhuma relação significativa.

Equação 5 - Probabilidade de erro

$$P(y|\theta, \sum M_i, t_h) \tag{5}$$

Fonte: O autor (2020)

A Equação 5 traz uma proposta de relação entre o alvo y a ser previsto e as variáveis, onde a letra grega theta minúscula representa a análise vetorial do corpo da mensagem, **M** é o nível de criticidade da mensagem, aqui apresentado como uma somatória de todos os níveis, e **t** é uma janela de tempo pré-determinada que, no caso desta proposta, será de 24 horas.

Tratar as mensagens de nível de criticidade e tempo é relativamente simples (a análise vetorial do corpo da mensagem foi apresentada no capítulo anterior). Como tanto o nível mensagem, quanto a data/horário ficam em uma posição fixa na linha de log, é possível extraí-las com uma rotina python de segmentação (*split*) simples (Figura 11)

Figura 11 - Segmentação da mensagem de log

```
log = ["Jan 1 00:06:13 root user.info mounting ext2 file system\
       using the ext4 subsystemmounting ext2 file system using\
       the ext4 subsystem"]
msg_split = [row.split(" ") for row in log]
date = msg_split[0][2]
message = msg_split[0][4]
print(date, '\t', end='')
print(message)
```

```
00:06:13        user.info
```

Fonte: O autor (2020)

A Figura 11 demonstra como fazer a segmentação e destacar os valores procurados, contudo ainda é preciso mapear o nível da mensagem, que precisa estar em caracteres numéricos, bem como obter apenas a hora em que a mensagem de log foi mostrada pois, conforme dito anteriormente, a ideia é trabalhar com uma janela de tempo de 24 horas e com as mensagens segmentadas por hora. A Figura 12 apresenta este tratamento.

Figura 12 - Convertendo os campos da mensagem

```
dicio = {'emer': [0], 'aler': [1], 'crit': [2], 'erro': [3],
         'warn': [4], 'noti': [5], 'info': [6], 'debu': [7],}
msg=dicio[message[5:]]
print(date[:2], '\t', end='')
print(msg)
```

```
00      [6]
```

Fonte: O autor (2020)

2.3.3 **Treinamento e Classificação**

Uma vez que tem-se os parâmetros definidos, conforme mostrado nos capítulos anteriores, pode-se pensar em como fazer o treinamento de um modelo baseado nisso. Os dados podem ser divididos entres unidades que não apresentaram falhas no disco rígido e aquelas que apresentaram.

Seguindo a linha de análise proposta, as unidades defeituosas devem ter seu log das últimas 24 horas antes do problemas ser verificado, analisado. Já para o cenário onde a unidade não apresenta nenhum problema, basta selecionar uma janela de 24 horas qualquer, pois o intuito será apenas analisar o comportamento em condições normais.

Tabela 2 - Proposta de conjunto de dados para treinamento

t	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	θ	y
0	0	0	0	5	4	8	12	1	1	1
1	0	0	0	1	3	2	19	1	1	1
2	0	0	1	3	1	17	8	1	1	1
[...]	[...]	[...]	[...]	[...]	[...]	[...]	[...]	[...]	[...]	
22	0	1	2	38	55	40	73	68	0	0
23	0	2	5	40	49	37	82	55	0	0

Fonte: O autor (2020)

A proposta é para que a cada intervalo de tempo t=1hora, seja contabilizada a quantidade mensagens de erro atribuídos a cada um dos níveis e também feita uma análise vetorial para capturar o "sentimento" do sistema (Tabela 2).

Os classificadores binários estão entre os mais comuns e bem compreendidos algorítimos no universo de aprendizagem de máquina E não exagero dizer que alguns deles como Random Forest, Naive Bayes e Redes Neurais representam o estado da arte quando o assunto é classificação por algorítimos supervisionados. Em outras palavras, algorítimos que trabalham com dados rotulados, tais quais os mostrados na Tabela 2.

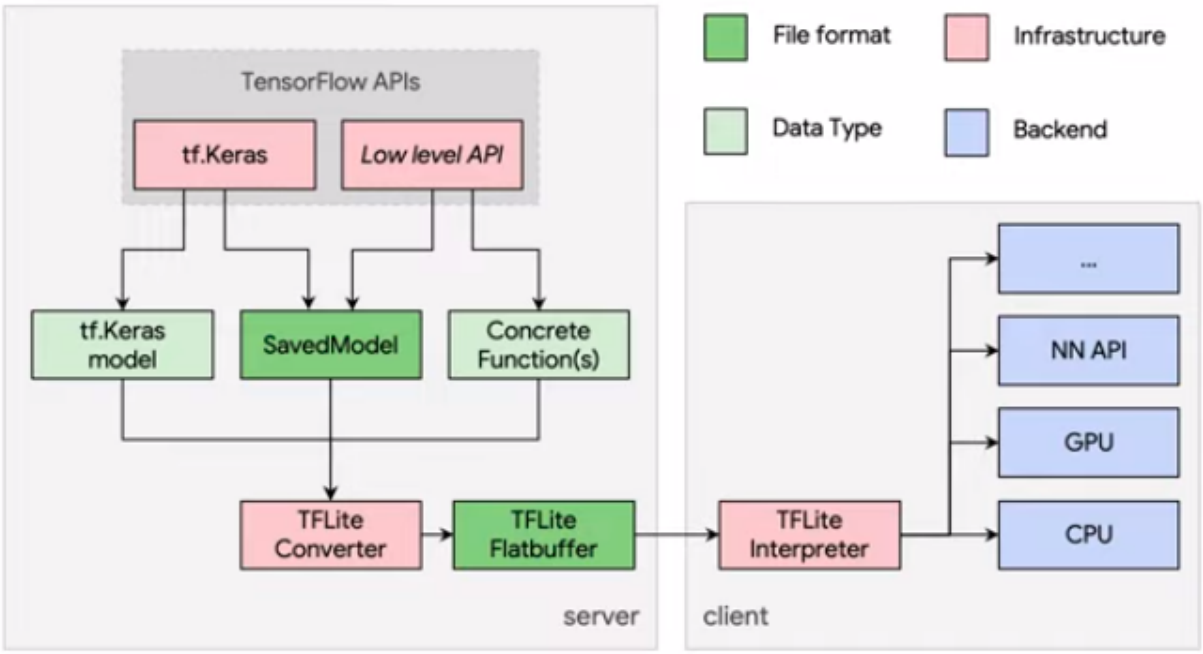
Uma vez que a proposta for implementada, a decisão de qual algorítimo implementar, dentre os que aqui foram citados, fica à cargo da melhor acurácia obtido com cada um deles.

2.3.4 **TensorFlow Lite - Embarcando o Modelo**

Na fase de tratamento e treinamento dos dados colhidos é natural que se utilize um computador com boa capacidade de processamento e, posteriormente, o modelo treinado deve ser integrado ao sistema embarcado. Mas como fazer essa integração? A resposta está no TensorFlow lite. Trata-se de um interpretador que pega um arquivo com o modelo treinado, executa as operações nos dados de entrada, e provê acesso à saída (TENSORFLOW, 2020).

Antes de ter integrar o modelo na plataforma de destino e realizar inferências, é preciso convertê-lo. Após treinar o modelo utilizando uma API de alto nível como o Keras, por exemplo, é preciso salvar o modelo e fazer a conversão para o modo *flatbuffer* (formato de serialização, otimizado para alta performance), utilizando conversor do TensorFlow Lite. Após a conversão, o arquivo é então salvo no formato .tflite, que é o formato reconhecido pela biblioteca que fará a interpretação do modelo na plataforma de destino. A Figura 13 exemplifica este processo

Figura 13 - Modelo TensorFlowLite



Fonte: Udacity (2019)

O TensorFlow lite é bastante versátil e possui suporte para Linux Embarcado. Sua biblioteca pode ser compilada tanto para a linguagem Python quanto para C++ e, no caso no caso da aproximação aqui proposta, não faz diferença qual das duas linguagens utilizar, pois uma vez que a biblioteca é compilada corretamente para o sistema embarcado em questão (arm 32 bits), deverá funcionar normalmente.

3 **CONCLUSÃO**

A proposta aqui apresentada visou mitigar o problema de análise de logs em um sistema complexo, onde uma grande quantidade de informações é gerada em intervalos de tempo relativamente pequeno, como é o caso de um sistema Linux, mais precisamente um Linux Embarcado.

A tarefa de tirar proveito de algorítimos de processamento de linguagem natural para análise destas informações geradas pelo sistema, aqui entendidas como mensagem de logs, mostra-se palpável, tendo em vista as diversas tecnologias disponíveis bem como uma vasta gama de estudos que buscam aplicar NLP fora de lugares comuns como análise de sentimento textuais de linguagem natural.

A linguagem empregada em um sistema Linux é uma linguagem complexa e, ainda mais para aquele que não está acostumado, seu entendimento é árduo. E é aí que surgem oportunidades e podem ser implementados classificadores para diversos fins.

Com tudo que foi estudado e apresentado neste trabalho, não resta dúvida que é possível implementar um sistema com o objetivo de alcançar o fim proposto de previsão de falhas em disco rígidos (ainda que sua acurácia seja imprevisível neste momento), possibilitando assim uma tomada de decisão preventiva. Tal decisão pode ser uma formatação de disco, a suspensão temporário do seu funcionamento ou até mesmo uma troca.

No futuro ainda é possível uma integração com tecnologia de monitoramento de disco rígido externo SMART (Self-Monitoring, Analysis and Reporting Technology, Tecnolologia de Auto-monitoramento, Análise e Relatório, em tradução livre para o português), sem contar o vasto e excitante universo da Inteligência Artificial que ainda pode ser explorado (ainda mais quando este que vos escreve acaba de descobri-lo).

REFERÊNCIAS

BERTERO, Christophe *et al*. **Experience Report**: Log Mining using Natural Language Processing and Application to Anomaly Detection. **LAAS CNRS**. Toulouse, France, 2017. 11 p. Disponível em: https://hal.laas.fr/hal-01576291/document. Acesso em: 3 Jul. 2020.

CARIN, Lawrence. **Words to Vectors**. **https://www.coursera.org/**. 2019. Disponível em: https://www.coursera.org/learn/machine-learning-duke/lecture/LUvrj/words-to-vectors. Acesso em: 1 Jul. 2020.

DEBUGGING by printing. **elinux**. 2019. Disponível em: https://elinux.org/Debugging_by_printing. Acesso em: 13 Jul. 2020.

MATOTEK, Dennis; TURNBULL, James; LIEVERDINK, Peter. **Pro Linux System Administration**: Learn to Build Systems for Your Business Using Free and Open Source Software. 2. ed. Nova York: Apress, v. 1, 2017.

RAO, Delip; MCMAHAN, Brian. **Natural Language Processing with PyTorch**: Build Inteligent Language Applications Using Deep Learning. 1. ed. California: O'Reilly, v. 1, 2019. 210 p.

TENSORFLOW. **Get started with TensorFlow Lite**. **TensorFlow**. 2020. Disponível em: https://www.tensorflow.org/lite/guide/get_started. Acesso em: 17 Jul. 2020.

UDACITY. **Introduction to TensorFlow Lite**: Learn how TensorFlow Lite works under the hood. **Udacity**. 2019. Disponível em: https://classroom.udacity.com/courses/ud190. Acesso em: 19 Jul. 2020.

YAGHMOUR, Karim *et al*. **Building Embedded Linux Systems**: Concepts, Techniques, Tricks & Traps. 2. ed. Sebastopol: O'Reilly, 2008.