Avaliação Comparativa de Técnicas baseadas em *Deep Learning* para Identificação de Duplicatas

Paulo Henrique Santos Lima, Wellington Santos Martins, Leonardo Andrade Ribeiro

[1]Instituto de Informática – Universidade Federal de Goiás(UFG) – Goiânia, GO – Brasil

pauloh@discente.ufg.br,{wellington,laribeiro}@inf.ufg.br

Abstract. *Application data inevitably has inconsistencies that may cause malfunctioning in daily operations and compromise analytical results. A particular type of inconsistency is the presence of duplicates, e.g., multiple and non-identical representations of the same information. Duplicate identification is challenging because they are not identical. Recently, DeepMatcher and Ditto, two solutions based on deep learning, have achieved state-of-the-art results in duplicate identification. However, DeepMatcher e Ditto did not consider duplicates with character-level variations in their experiments; such variations are pervasive in real-world databases. This paper presents a comparative evaluation between DeepMatcher and Ditto on data with textual patterns that were not considered in previous experiments. The results showed that the two solutions experienced a drop in accuracy, while Ditto was more robust than DeepMatcher.*

Resumo. *Dados de aplicações possuem inevitavelmente inconsistências que podem causar mal funcionamento em operacões rotineiras e compromenter resultados analíticos. Um tipo particular de inconsistência é a presença de duplicatas, isto é, múltiplas e não idênticas representações da mesma informação. A identificação de duplicatas é difícil porque elas não são cópias idênticas entre si. Recentemente, DeepMatcher e Ditto, duas soluções baseadas em deep learning, obtiveram resultados do estado da arte em identificação de duplicatas. Entretanto, DeepMatcher e Ditto não consideraram duplicatas com variações textuais em nível de caracteres em seus experimentos; tais variações são pervasivas em bancos de dados do mundo real. Este artigo apresenta uma avaliação comparativa do DeepMatcher e Ditto em dados com padrões textuais que não foram considerados nos experimentos anteriores. Os resultados obtidos demonstraram que as duas soluções experimentaram queda de acurácia, sendo que o Ditto apresentou maior robustez em comparação com DeepMatcher.*

1. Introdução

Um adágio conhecido na área de Banco de Dados diz que "dados reais são sujos" [Hernández and Stolfo 1998]. Em outras palavras, dados gerados e usados por aplicações apresentam inevitavelmente inconsistências, como violações de padrões e restrições de integridade, presença de valores atípicos (*outliers*) e representações diferentes de um mesmo objeto de informação [Abedjan et al. 2016]. Além de causar mal funcionamento em operações rotineiras de uma organização, como venda de produtos e controle de estoque, a presença dessas inconsistências também pode comprometer processos de análise de dados. Portanto, a identificação e correção de inconsistências é uma tarefa fundamental em sistemas centrados em dados.

	Título	Autor	Editora
t_1	O Hobbit	J. R. R. Tolkien	WMF Martins Fontes
t_2	O Hobbit	Tolkien	Martins Fontes
t_3	O Jobbit	J. R. R. Token	WMF Martim Fontes

(a) Dados estruturados

	Descrição
t_1	Fone de Ouvido JBL Quantum100 Headset Gamer Preto
t_2	Fone Headset Gamer JBL Quantum 100 - Drivers 40mm - Preto
t_3	Fone Headset Gamer JBK Quntum 100 - Petro

(b) Dados textuais.

	Produto	Empresa	Preço
t_1	Office	Microsoft	450.00
t_2	Microsoft Office		450.00
t_3	Micosoft Offive		450.00

(c) Dados sujos.

Figura 1. Tipos de dados considerados neste artigo.

Entre as inconsistências supracitadas, a identificação de múltiplas representações de uma mesma entidade do mundo real é de particular importância. Essas representações redundantes são chamadas frequentemente de duplicatas difusas [Ananthakrishna et al. 2002], por não serem cópias idênticas entre si, ou simplesmente duplicatas [de Oliveira and Ribeiro 2019]. A presença de duplicatas em um banco de dados pode ser causada por vários motivos, possivelmente interrelacionados; entre os mais comuns, estão erros durante a entrada ou geração de dados, diferentes convenções, nomenclaturas divergentes e integração de fontes de dados independentes contendo informações em comum. A Figura 1 ilustra diferentes tipos de duplicatas em bancos de dados relacionais. O problema da identificação de duplicatas (ID) possui um longo histórico de pesquisa. De fato, os primeiros estudos neste tema foram realizada ainda na década de 1950 [Newcombe et al. 1959]. Apesar dos intensos esforços ao longo dos anos, ID ainda é um problema em aberto, em particular quando aplicada sobre grandes volumes de dados [Stonebraker and Ilyas 2018].

Nos últimos anos, *deep learning* (DL) tem promovido significativos avanços em aprendizado de máquina e inteligência artificial [Krizhevsky et al. 2012]. Mais especificamente, DL tem obtido resultados do estado da arte em tarefas sobre dados que possuem algum tipo de estrutura implícita, como texto, imagem e voz. Usando um conjunto de exemplos (dados rotulados), DL constrói automaticamente importantes características dos dados, evitando a necessidade de intervenção manual (*feature engineering*). Com isso, técnicas baseadas em DL geraram grande impacto em áreas como processamento de imagens, robótica, diagnóstico médico, processamento de linguagem natural (PLN), entre outras [LeCun et al. 2015].

Recentemente, DL tem sido também investigado para resolver o problema de identificação de duplicatas [Ebraheem et al. 2018, Mudgal et al. 2018, Brunner and Stockinger 2020, Li et al. 2020, Barlaug and Gulla 2021]. O trabalho em [Mudgal et al. 2018] apresentou *DeepMatcher*, uma solução baseada em rede neural recorrente (RNR) e mecanismo de atenção. *DeepMatcher* obteve ganhos significativos de acurácia em comparação com *Magellan* [Konda et al. 2016], uma solução até então do estado da arte baseada em modelos de aprendizagem. Uma outra abordagem para ID usa modelos de linguagem pré-treinados baseados na arquitetura *Transformer* [Vaswani et al. 2017]. O trabalho em [Li et al. 2020] apresentou *Ditto*, uma solução que utiliza esses modelos pré-treinados acoplados a uma camada customizada para ID visando alavancar transferência de aprendizagem (*transfer learning*). *Ditto* superou *DeepMatcher* em todos cenários analisados, em particular quando dados de trainamento são escassos.

DeepMatcher e *Ditto* consideraram três tipos de dados de entrada: estruturado, textual e ”sujo”. Dados estruturados possuem um esquema rígido, com valores simples e

atômicos, como *strings* curtas e números (Figura 1(a)). Dados textuais são caracterizados por textos longos como descrições de produtos, textos obtidos de páginas Web e conteúdo de redes sociais (Figura 1(b)). Finalmente, dados sujos também possuem um esquema rígido como dados estruturados, mas apresentam atributos com valores ausentes ou referentes a outros atributos. Esta situação é comum quando dados estruturados são obtidos a partir de um processo de extração de informação. Por exemplo, o nome de uma empresa pode ser inadvertidamente inserido no atributo `Produto` (Figura 1(c)).

Entretanto, *DeepMatcher* e *Ditto* não consideraram dados com variações textuais em nível de caracteres. Tais variações são pervasivas em bancos de dados reais, sendo causadas por erros durante a ingestão manual de dados, como erros de digitação, ou ingestão automatizada, como falhas em processos de digitalização. Esses erros podem introduzir *tokens* não contemplados nos dados de treinamento (*out-of-vocabulary*) e degradar, por exemplo, a eficácia de abordagens baseadas em modelos pré-treinados. Note que os dados sujos mencionados anteriormente contém apenas a transposição de valores entre atributos, mas não variações nos valores dos atributos. Na Figura 1, tuplas `t1` e `t2` nos três tipos de dados exemplificam as instâncias consideradas por *DeepMatcher* e *Ditto*, ao passo que tupla `t3` ilustra instâncias com a presença de possíveis erros de digitação.

Este artigo apresenta uma avaliação do *DeepMatcher* e *Ditto* em dados com variações textuais em nível de caracteres. Para isso, serão utilizados os conjuntos de dados considerados nesses trabalhos, mas com a injeção aleátoria de modificações textuais em diferentes proporções. O objetivo dessa avaliação é responder as seguintes questões: 1) quão robusto são *DeepMatcher* e *Ditto* em dados com as variações supracitadas?; 2) como essas variações afetam o desempenho dessas soluções em cada tipo de dados? e 3) como essas variações afetam comparativamente *DeepMatcher* e *Ditto*.

O restante deste artigo está organizado como descrito a seguir. Seção 2 apresenta o referencial teórico. Seção 3 apresenta e analisa os resultados experimentais. Os trabalhos relacionados são discutidos na Seção 4. Finalmente, Seção 5 apresenta as conclusões e delineia trabalhos futuros.

2. Referencial Teórico

Esta seção apresenta primeiramente a definição formal do problema. Em seguida, serão apresentados detalhes do *DeepMatcher* e *Ditto*.

2.1. Definição do Problema

Sejam D e D' duas fontes de dados que possuem um mesmo esquema com os atributos $A_1, ..., A_N$. Nas duas fontes, cada tupla representa uma entidade do mundo real, podendo essa entidade ser um objeto físico, abstrato ou conceitual. O objetivo do processo de identificação de duplicatas é encontrar a maior relação binária $M \in D \times D'$, em que cada par $(e_1, e_2) \in M, e_1 \in D, e_2 \in D'$, representa a mesma entidade; tem-se $D = D'$, caso o objetivo seja encontrar duplicatas em uma única fonte de dados. Assume-se a disponibilidade de um conjunto de dados de treinamento $T \in M$ de tuplas $\{(e_1^i, e_2^i, r)\}_{i=1}^{|T|}$, onde $\{(e_1^i, e_2^i)\}_{i=1}^{|T|} \subseteq M$ e r é um rótulo com valores em {"*duplicata*", "*não duplicata*"}. Neste contexto, a partir de T, uma solução baseada em DL visa construir um classificador que possa corretamente distinguir pares de tuplas entre "duplicata"e "não duplicata".

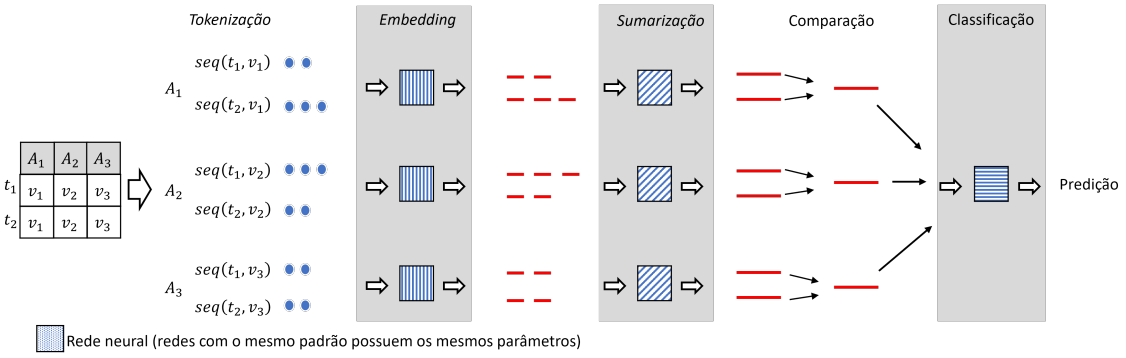

Figura 2. *Template* arquitetural do DeepMatcher.

2.2. *DeepMatcher*

DeepMatcher é baseado em um *template* arquitetural, ilustrado na Figura 2, que permite construir um rico espaço de soluções. Os dados de entrada são constituídos por pares de tuplas representando possíveis duplicatas. Na etapa de *tokenização*, os valores textuais de cada atributo das duas tuplas são segmentados em sequências de palavras (os termos *token* e palavra são usados de maneira intercambiável neste artigo). Nas etapas seguintes, DL pode ser empregada de diferentes maneiras.

Na etapa de *embedding*, as sequências de palavras são representadas como sequências de vetores númericos. Abordagens possíveis para esta etapa podem ser definidas ao longo de dois eixos: granularidade do *embedding* e uso ou não de modelos pré-treinados. A granularidade do *embedding* pode ser a nível de palavras ou caracteres. No primeiro caso, uma tabela é aprendida mapeando cada palavra para um valor númerico. No segundo caso, um modelo é treinado para produzir *embeddings* para palavras contendo os caracteres que compõem o vocabulário do modelo. *Embeddings* a nível de caracteres são mais robustos em lidar com palavras infrequentes ou *out-of-vocabulary*; essas últimas podem ser causadas por erros de digitação, como mencionado anteriormente. Para essas duas granularidades, modelos pré-treinados podem ser utilizados ou todo treinamento pode ser realizado do ínicio. A última opção pode ser preferível em domínios que contém *tokens* com semântica específica, como códigos de produtos.

Em sumarização, as sequências de vetores representando os atributo de cada tupla são agregadas em um único vetor. Para isso, RNRs podem ser usadas para capturar a ordem e a semântica da sequência de *tokens*. De maneira geral, RNRs têm dificuldade em representar longas sequências de tokens. Outra limitação dessa abordagem é que as duas sequências não são consideradas conjuntamente no processo de sumarização. Com isso, a similaridade subjacente entre sequências de diferentes tamanhos pode não ser capturada. Uma abordagem diferente consiste em usar um mecanismo de atenção para que a sequência de vetores oriundos de uma tupla seja usada como contexto no processo de sumarização da sequência de vetores da outra tupla. Entretanto, informação sobre a posição dos tokens de entrada será perdida no processo de sumarização. Essa informação é importante pois é comum casos em que, por exemplo, o token mais relevante é o primeiro. Finalmente, é possível adotar uma estratégia híbrida, combinando RNRs e mecanismos de atenção para obter-se as vantagens e evitar os problemas de cada abordagem, ao custo de um aumento da complexidade do modelo de aprendizagem.

Figura 3. *Pipeline* de pré-processamento do *Ditto*.

Em comparação, a similaridade dos vetores sumarizados é aferida usando, por exemplo, a função cosseno. O resultado da comparação de cada atributo é um valor de similaridade escalar; esses valores comporão o conjunto de características da rede neural usada na etapa de classificação. Outra possibilidade é usar como função de comparação operações como concatenação e diferença absoluta entre elementos e relegar ao classificador a tarefa de aprender uma função de similaridade. Finalmente, a saída do classificador determinará se o par de tuplas de entrada representa a mesma entidade.

Na avaliação empírica de [Mudgal et al. 2018], *DeepMatcher* obteve resultados em dados estruturados similares a modelos de aprendizagem mais simples, como florestas aleatórias (*random forests*) e regressão logística, que demandam tempo de treinamento bem menor. Por outro lado, ganhos expressivos de acurácia foram obtidos em dados textuais e sujos. Além disso, diversas instâncias do template arquitetural da Figura 2 foram avaliadas. A configuração que obteve os melhores resultados utilizou *fastText* [Bojanowski et al. 2017], um modelo de *embedding* pré-treinado em nível de caracter, sumarização combinando RNR com mecanismo de atenção, função de similaridade aprendida pela rede neural *perceptron* multicamada usada como classificador. Essa será a configuração do *DeepMatcher* avalidada neste artigo.

2.3. *Ditto*

A adoção de modelos de linguagem baseados na arquitetura *Transformer* permitiu avançar ainda mais o estado da arte em soluções para ID [Brunner and Stockinger 2020, Li et al. 2020]. Esses modelos são treinados usando corpus volumosos de texto, como a Wikipedia, de maneira não supervisionada. A arquitetura *Transformer* substitui completamente RNRs por um mecanismo de auto-atenção [Vaswani et al. 2017] que permite gerar *embeddings* de *tokens* considerando todos os demais *tokens* da sequência de entrada. Com isso, esses *embeddings* capturam informações semânticas e contextuais, incluindo aspectos linguísticos intrincados como polissemia e sinonímia. BERT [Devlin et al. 2019] é o modelo de linguagem pré-treinado baseado em *Transformers* mais popular.

Ditto aplica um ajuste fino nesses modelos pré-treinados para a tarefa de ID. Para isso, são adicionadas uma camada completamente conectada e uma camada de saída que usa a função $softmax$. A rede modificada é então inicializada com os parâmetros do modelo pré-treinado e treinada posteriormente com os dados em T até convergir. Além desse ajuste fino, *Ditto* usa um método para serializar os pares de entrada em uma sequência de *tokens* e realiza três otimizações na etapa de pré-processamento: inserção de conhecimento de domínio, sumarização de sequências longas e aumento dos dados de treino com exemplos difíceis. Figura 3 ilustra essas etapas de pré-processamento.

Ditto serializa uma tupla da seguinte maneira:

$$serialize(t) = [COL]A_1[VAL]v_1...[COL]A_N[VAL]v_N,$$

onde `[COL]` e `[VAL]` são *tokens* especiais indicando o ínicio de nomes de atributos e valores, respectivamente. Por exemplo, o resultado da serialização da tupla t_2

na Figura 1(c) é dado por: `[COL]` *Produto* `[VAL]` *Microsoft Office* `[COL]` *Empresa* `[VAL]` *NULL* `[COL]` *Preço* `[VAL]` *450,00*.

Finalmente, pares de tuplas são serializados da seguinte maneira:

$$serialize(t_1, t_2) = [CLS]serialize(t_1)[SEP]serialize(t_2)[SEP],$$

onde `[SEP]` é o *token* especial usado para delimitar as representações de t_1 e t_2 e `[CLS]` é o *token* especial cujo *embedding* associado representará a codificação do par de tuplas.

Conhecimento de domínio pode ser inserido nas entradas serializadas de duas maneiras: inclusão de *tokens* especiais identificando trechos com semântica específica como códigos de produtos e números de ruas; e uniformização de sinônimos em uma única *string*. Sumarização reduz sequências longas na entrada serializada para o tamanho máximo permitido pelo BERT que é 512 *tokens*. Finalmente, aumento dos dados é realizado via operadores que geram novos pares de entrada serializados a partir de pares existentes. Esses operadores aplicam modificações aleatórias nos pares originais como exclusão e transposição de tokens.

Em contraste com *DeepMatcher*, *Ditto* não requer que os dados de entrada tenham o mesmo esquema. O método de serialização permite, inclusive, aplicar *Ditto* em dados hierárquicos como XML e JSON usando *tokens* especiais para representação de pares atributo-valor aninhados. Outra diferença com *DeepMatcher* é que em *Ditto* o mecanismo de atenção cruzada entre o par de tuplas não é limitado a palavras de um mesmo atributo. Essas diferenças podem ser atenuadas aplicando o método de serialização também no *DeepMatcher* e associando o resultado a um único atributo — a avaliação dessa estratégia é deixada para trabalhos futuros. De qualquer maneira, *Ditto* ainda possui uma arquitetura mais simples que o *DeepMatcher*, onde os componentes de *embedding*, sumarização e comparação (ver Figura 2) são substituídos por um modelo de linguagem pré-treinado.

Finalmente, *Ditto* possui suporte a outros modelos de linguagem além do BERT. O modelo que apresentou os melhores resultados foi o RoBERTa [Liu et al. 2019], uma variante do BERT com um conjunto de melhorias e que foi treinada em volume maior de dados. Esse será o modelo considerado nos experimentos.

3. Experimentos

Esta seção apresenta a avaliação empírica realizada, cujo objetivo foi avaliar e comparar o desempenho do *DeepMatcher* e *Ditto* em dados com variações textuais em nível de caracteres. Primeiro, serão descritos os conjuntos de dados, configuração dos métodos analisados, hardware e software utilizados e métricas usadas na comparação. Em seguida, os resultados serão apresentados e discutidos.

3.1. Descrição do Ambiente Experimental

Os experimentos usaram os mesmos *datasets* públicos da avaliação do *DeepMatcher Ditto*. Detalhes sobre esses *datasets* são apresentados na Tabela 1. Os *datasets* são provenientes de um variedade de domínios e possuem diferentes características, sendo cinco estruturados, um textual e dois sujos; esses últimos foram gerados a partir dos *datasets* estruturados através da transposição de valores entre atributos como mencionado na Seção 1. Em todos os *datasets*, cada item de dados é composto por um par de tuplas e um rótulo

Tabela 1. *Datasets* usados nos experimentos.

Tipo	Dataset	Domínio	Tamanho	# Positivos	# Atributos
Estruturado	Amazon-Google	softwares	11.460	1.167	3
	BeerAdvo-RateBeer	bebidas	450	68	4
	DBLP-ACM	citações	12.363	2.220	4
	DBLP-Scholar	citações	28.707	5.347	4
	Walmart-Amazon	eletrônicos	10.242	962	5
Textual	Abt-Buy	produtos	9.575	1.028	3
Sujo	DBLP-ACM	citações	12.363	2.220	4
	DBLP-Scholar	citações	28.707	5.347	4

Tabela 2. Descrição das modificações realizadas nos *datasets* originais.

Nome	E1	E2	E3	E4
% erros	90%	50%	30%	10%
# carac. mod.	1-2	1-2	3-5	3-5
# atributos	2	2	2	2

que classifica esse par como duplicata ou não duplicata. O tamanho dos *datasets*, quantidade de pares classificados como duplicatas e quantidade de atributos são informados nas colunas 4–6, respectivamente.

Para cada *dataset*, foram geradas quatro cópias com diferentes proporções de pares de tuplas alterados e, em cada par alterado, diferentes quantidades de caracteres afetados pelas modificações de inserção, exclusão ou substituição. A Tabela 2 descreve essas alterações. Por exemplo, na cópia `E1`, 90% dos pares de tuplas tiveram 1–2 caracteres modificados. Cada par modificado foi selecionado aleatoriamente no *dataset* original, assim como o caracter e o tipo de cada modificação. Com isso, as modificações podem afetar as duas tuplas de cada par ou apenas uma delas. Para cada *dataset*, foram escolhidos os dois atributos com valores considerados mais informativos para serem modificados. Por exemplo, em *DBLP-ACM*, foram modificados os atributos `titulo` e `autor`. Cada par de tuplas modificado substitui o par original e os rótulos não são alterados. Portanto, as cópias E1-4 possuem estatísticas dos *datasets* originais apresentadas na Tabela 1. Finalmente, cada *dataset* foi dividido em conjuntos de treino, validação e teste na proporção de 3:1:1, seguindo a mesma divisão usada nos artigos do *DeepMatcher* e *Ditto*.

Como mencionado, foram avaliadas as versões das soluções que apresentaram os melhores resultados nos artigos originais: para o *DeepMatcher* foi avaliada a versão *Hybrid*, que utiliza a estratégia mais complexa de sumarização e para o *Ditto* , foi a avaliada a versão que emprega todas as otimização e o modelo RoBERTa. Os experimentos usaram as implementações do *DeepMatcher*[1] e *Ditto* [2] disponibilizadas publicamente pelos autores. *DeepMatcher* foi implementado usando o *Torch*[3], um *framework* com suporte ao uso de GPUs para acelerar o treinamento. *Ditto* é implementado usando *PyTorch* [4] e

[1]https://github.com/anhaidgroup/deepmatcher. Último acesso em 25.05.2022
[2]https://github.com/megagonlabs/ditto. Último acesso em 25.05.2022.
[3]http://torch.ch/. Último acesso em 07.07.2022
[4]https://pytorch.org. Último acesso em 07.07.2022.

Tabela 3. Resultados de acurácia em dados estruturados.

dataset/técnica		Original	E1	ΔF_1	E2	ΔF_1	E3	ΔF_1	E4	ΔF_1
Amazon-Google	**DeepMatcher**	68.94	40.60	-28.34	56.07	-12.87	47.20	-21.74	54.74	-14.20
	Ditto	71.58	57.14	-14.44	68.79	-2.79	63.72	-7.86	18.51	-53.07
Beer	**DeepMatcher**	70.97	64.52	-6.45	70.97	0	62.86	-8.11	68.75	-2.22
	Ditto	85.71	55.17	-30.54	66.66	-19.05	68.96	-16.75	66.66	-19.05
DBLP-ACM	**DeepMatcher**	98.76	98.43	-0.33	98.31	-0.45	98.00	-0.76	98.77	0.01
	Ditto	98.00	98.42	0.42	98.53	0.53	97.83	-0.17	98.52	0.52
DBLP-Google	**DeepMatcher**	94.90	91.80	-3.10	93.51	-1.39	92.56	-2.34	93.29	-1.61
	Ditto	95.05	94.00	-1.05	94.56	-0.49	94.38	-0.67	94.17	-0.88
Walmart-Amazon	**DeepMatcher**	63.66	60.57	-3.09	62.32	-1.34	61.28	-2.38	62.05	-1.61
	Ditto	85.93	81.52	-4.41	80.40	-5.53	29.08	-56.85	84.55	-1.38

Tabela 4. Resultados de acurácia em dados textuais.

dataset/técnica		Original	E1	ΔF_1	E2	ΔF_1	E3	ΔF_1	E4	ΔF_1
Abt-Buy	**DeepMatcher**	70.03	56.31	-13.72	61.63	-8.40	64.66	-0.37	65.96	-4.07
	Ditto	89.15	24.74	-64.41	88.05	-1.10	78.46	-10.69	84.93	-4.22

Hugging Face Transformers [5]. Para o *Ditto*, foi usada a otimização de ponto flutuante de meia precisão (*fp16*) para acelerar o treinamento e teste, o tamanho máximo da sequência de entrada foi fixado em 256, a taxa de aprendizado em *3e-5* de modo linearmente decrescente e o tamanho do lote foi fixado em 32. Todos experimentos foram realizados através do *Google Collaboratory*[6], que permite ao usuário executar *Python* pelo navegador e utilizar GPUs gratuitamente. Por causa das limitações de recursos computacionais disponíveis, os treinamentos foram realizados em 15 épocas. Outros parâmetros foram definidos de acordo com configurações descritas em cada artigo.

Os resultados de acurácia são reportados pela métrica F1, que é a média harmônica entre precisão e revocação. Dado o resultado do teste de um modelo, seja VP a quantidade de pares corretamente classificados como duplicatas, FN a quantidde de pares incorretamente classicados como duplicatas e FN a quantidade de pares incorretamente classificados como não duplicatas. A precisão P é dada por $P = VP/(VP+FP)$ e a revocação R é dado por $R = VP/(VP + FN)$. Portanto, $F1$ é dado por $2 \times ((P \times R)/(P + R))$.

3.2. Resultados e Discussão

Os resultados de acurácia do *DeepMatcher* e *Ditto* em dados estruturados, textuais e sujos são apresentados nas Tabelas 3, 4, 5, respectivamente. Para cada *dataset*, são reportados os resultados em sua versão original, sem modificações, e nas versões E1–4 com modificações textuais em nível de caracteres; os resultados de cada versão modificada são acompanhados com a sua diferença em relação ao resultado obtido na versão original. O melhor resulta obtido em cada *dataset* está destacado em vermelho.

Em relação aos resultados obtidos nos *datasets* originais, os mesmos seguiram as mesmas tendências da comparação realizada em [Li et al. 2020], com *Ditto* apresentando

[5]https://huggingface.co. Último acesso em 07.07.2022.

[6]https://colab.research.google.com. Último acesso em 07.07.2022

Tabela 5. Resultados de acurácia em dados sujos.

dataset/técnica		Original	E1	ΔF_1	E2	ΔF_1	E3	ΔF_1	E4	ΔF_1
DBLP-ACM	**DeepMatcher**	97.20	93.41	-3.79	95.51	-1.69	94.53	-2.67	92.63	-4.57
	Ditto	97.62	98.08	0.46	97.87	0.25	97.96	0.34	98.30	0.68
DBLP-Google	**DeepMatcher**	92.22	90.32	-1.90	91.26	-0.96	90.39	-1.83	91.49	-0.73
	Ditto	95.07	94.06	-1.01	94.69	-0.38	94.44	-0.63	95.04	-0.03

um clara vantagem sobre o *DeepMatcher*. Nos dados estruturados, *Ditto* é superior em 3 dos 5 *datasets* e comparável nos demais; a maior vantagem em acurácia é de 22% no *dataset Walmart-Amazon*. *Ditto* também é superior ao *DeepMatcher* no *dataset* textual com vantagem de 19% na métria F1. Nos *datasets* sujos os resultados do *Ditto* são próximos aos do *DeepMatcher*, mas ainda superiores.

Em relação aos resultados obtidos nos *datasets* modificados, as duas soluções experimentaram queda de acurácia em quase todos casos. A maior queda de acurácia em média ocorreu no *dataset Amazon-Google*. Este é o *dataset* considerado mais difícil para a tarefa de ID, isto é, a distinção entre duplicatas e não duplicatas é mais tênue; note que este é o *dataset* no qual as duas soluções obtiveram o pior resultado na sua versão original. Por outro lado, as exceções em que a acurácia aumentou nas versões modificadas concentram-se no *dataset DBLP-ACM*, tanto na versão estruturada quanta na suja. Este é o *dataset* considerado mais fácil para a tarefa de ID, isto é, existe uma separação mais clara entre duplicatas e não duplicatas; note que este é o *dataset* no qual as duas soluções obtiveram o melhor resultado na sua versão original. *Ditto* apresenta ligeiros ganhos de acurácia na maioria das versões modificadas do *DBLP-ACM*, com maior ganho em 2.3%. É possível concluir que o efeito das modificações textuais acompanha as características dos *datasets* originais em termos da separação entre duplicatas e não duplicatas: o efeito dessas modificações é negligível em *datasets* fáceis e significativo nos difíceis.

Nos dados estruturados, o *Ditto* apresenta um desempenho superior em quase todas as situações. Exceções ocorreram nos seguintes casos: *dataset Beer*, no qual o *Ditto* ele é ligeiramente inferior ao *DeepMatcher*; e conjuntos E4 do *Amazon-Google* e E3 do *Walmart-Amazon*, por causa de problemas ainda não identificados durante geração do modelo que causaram valores *outliers* nos resultados. Nos dados textuais, o *Ditto* supera o *DeepMatcher*, porém apresenta um valor *outlier* no conjunto E1. Por fim, nos dados sujos, o *Ditto* é superior ao *DeepMatcher* em todos os casos.

Analisando o desempenho individual de cada modelo se observa que o *DeepMatcher* tem uma queda de desempenho em quase todos os *datasets* no conjunto E1, que contém 90% dos pares alterados em 1 ou 2 caracteres. Em seguida, o fator de maior queda foi a quantidade de erros, assim se tem o conjunto E4 (10% 3 a 5), E3 (30% 3 a 5) e o E2 (50% 1 a 2), respectivamente. Por fim, no *Ditto*, o padrão não é tão claro mas se nota um padrão similar do *DeepMatcher* nos tipos estruturado e textual. Porém no sujo há uma mudança: o pior caso foi influenciado pela porcentagem de erros, assim o conjunto com pior de desempenho foi E1, E2, E3 e E4, respectivamente.

A Tabela 6 apresenta o tempo de treinamento de cada modelo nos *datasets* originais. *Ditto* apresenta um tempo maior para realizar o treinamento nos três *datasets*: nos dados estruturados, o tempo total do *DeepMatcher* corresponde a 36.69% do tempo to-

Tabela 6. Comparação entre o tempo de treinamento nos três tipos de dados.

técnica/dataset	Amazon-Google	Beer	DBLP-ACM	DBLP-Google	Walmart-Amazon	Total
DeepMatcher	0:07:08	00:00:28	00:12:12	00:27:08	00:10:30	0:57:26
Ditto	0:19:17	0:02:14	0:41:02	1:08:02	0:25:57	2:36:32

(a) Dados estruturados.

técnica/dataset	Abt-Buy
DeepMatcher	00:09:11
Ditto	00:38:07

(b) Dados textuais.

técnica/dataset	DBLP-ACM	DBLP-Google	Total
DeepMatcher	0:14:28	0:30:41	0:45:09
Ditto	0:41:36	1:08:34	1:50:10

(c) Dados sujos.

tal do *Ditto* para realizar o treinamento (Tabela 6(a)); nos dados textuais, o tempo de treinamento do *DeepMatcher* corresponde a 24.09% do tempo total do *Ditto* (Tabela 6(b)); finalmente, nos dados sujos, o tempo do *DeepMatcher* é 40.98% do tempo do *Ditto* (Tabela 6(c)). É importante ressaltar que estes valores são apenas indicativos no contexto de um ambiente de processamento compartilhado como *Google Colab*. Para uma comparação precisa, seria necessário o uso de *hardware* dedicado que não estava dispónivel para realização desse experimento.

4. Trabalhos Relacionados

ID vem sendo estudado desde o final da década de 1950, a partir do trabalho pioneiro em [Newcombe et al. 1959]. Desde então, várias comunidades científicas, incluindo Banco de Dados, Recuperação de Informação, PLN, Aprendizagem de Máquina, Web Semântica e Estatística, têm abordado diferentes aspectos do problema, frequentemente usando diferentes termos como resolução de entidades, casamento de registros, deduplicação e reconciliação de referências. Uma revisão da litetura anterior à emergência de DL é apresentada em [Elmagarmid et al. 2007]. Outra revisão, mais recente e com foco em técnicas baseadas em DL, é apresentada em [Barlaug and Gulla 2021]. *DeepMatcher* e *Ditto* são exemplos representativos dentre o conjunto atual de soluções baseadas em DL. Por exemplo, *DeepER* [Ebraheem et al. 2018] corresponde a instâncias mais simples do *template* arquitetural do *DeepMatcher*, ao passo que a solução em [Brunner and Stockinger 2020] corresponde a versão básica do *Ditto* sem a aplicação de otimizações.

ID possui estreito relacionamento com diversos outros problemas em PLN e integração de dados, frequentemente com soluções intercambiáveis. Alguns exemplos desses problemas incluem: ligação de entidades [Shen et al. 2015], que visa ligar menções a entidades em um documento a uma entidade representada em uma base de conhecimento; alinhamento de entidades [Leone et al. 2022] que visa encontrar equivalências entre entidades de duas bases de conhecimento diferentes; e resolução de coreferência [Clark and Manning 2016], que visa identificar segmentos de texto em documentos que se referem a uma mesma entidade.

5. Conclusões e Trabalhos Futuros

Recentemente, soluções baseadas *deep learning* tem obtido resultados do estado da arte para o problema da identificação de duplicatas. Entre essas soluções, destacam-se *DeepMatcher*, que adota uma abordagem baseada em redes neurais recorrentes e mecanismos de atenção, e *Ditto*, que utiliza modelos de linguagem pré-treinados baseados na arquitetura *Transformer*. Entretanto, *DeepMatcher* e *Ditto* não consideraram em seus experi-

mentos duplicatas com variações textuais em nível de caracteres; tais variações ocorrem, por exemplo, devido a erros de digitação, e são frequentemente encontradas em bancos de dados do mundo real. Neste contexto, este artigo apresentou uma avaliação comparativa do *DeepMatcher* e *Ditto* em dados com padrões textuais não considerados nos experimentos anteriores. Os resultados obtidos mostraram que as duas soluções experimentaram queda de acurácia na maioria dos cenários analisados, sendo que *Ditto* apresentou, de maneira geral, maior eficácia e robustez em comparação com *DeepMatcher* ao custo de exigir um maior tempo de treinamento. Além disso, observou-se que o efeito das modificações textuais no resultado da classificação é influenciado pelas características dos *datasets* originais em termos da separação entre duplicatas e não duplicatas: o efeito dessas modificações é negligível em *datasets* de fácil separação e significativo nos *datasets* de difícil separação. Trabalhos futuros incluem o estudo de melhorias no processo de treinamento para capturar as modificações textuais consideradas neste artigo e experimentos com diferentes representações de dados.

Agradecimento

Paulo Henrique Santos Lima foi estudante do Programa de Iniciação à Pesquisa da UFG com bolsa concedida pelo CNPq.

Referências

Abedjan, Z., Chu, X., Deng, D., Fernandez, R. C., Ilyas, I. F., Ouzzani, M., Papotti, P., Stonebraker, M., and Tang, N. (2016). Detecting Data Errors: Where Are We and What Needs to Be Done? *Proceedings of the VLDB Endowment*, 9(12):993–1004.

Ananthakrishna, R., Chaudhuri, S., and Ganti, V. (2002). Eliminating Fuzzy Duplicates in Data Warehouses. In *Proceedings of the VLDB Conference*, pages 586–597.

Barlaug, N. and Gulla, J. A. (2021). Neural Networks for Entity Matching: A Survey. *ACM Transactions on Knowledge Discovery from Data*, 15(3):52:1–52:37.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Brunner, U. and Stockinger, K. (2020). Entity Matching with Transformer Architectures – A Step Forward in Data Integration. In *Proceedings of the International Conference on Extending Database Technology*, pages 463–473.

Clark, K. and Manning, C. D. (2016). Improving Coreference Resolution by Learning Entity-Level Distributed Representations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 643–653.

de Oliveira, J. D. and Ribeiro, L. A. (2019). Avaliação de Processos ETL para Análise de Dados usando SGBD Orientado a Grafos. In *Anais da VII Escola Regional de Informática de Goiás*, pages 61–74.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

Ebraheem, M., Thirumuruganathan, S., Joty, S. R., Ouzzani, M., and Tang, N. (2018). Distributed Representations of Tuples for Entity Resolution. *Proceedings of the VLDB Endowment*, 11(11):1454–1467.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16.

Hernández, M. A. and Stolfo, S. J. (1998). Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37.

Konda, P., Das, S., C., P. S. G., Doan, A., Ardalan, A., Ballard, J. R., Li, H., Panahi, F., Zhang, H., Naughton, J. F., Prasad, S., Krishnan, G., Deep, R., and Raghavendra, V. (2016). Magellan: Toward Building Entity Matching Management Systems. *Proceedings of the VLDB Endowment*, 9(12):1197–1208.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 1106–1114.

LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep Learning. *Nature*, 521(7553):436–444.

Leone, M., Huber, S., Arora, A., García-Durán, A., and West, R. (2022). A Critical Re-evaluation of Neural Methods for Entity Alignment. *Proceedings of the VLDB Endowment*, 15(8):1712–1725.

Li, Y., Li, J., Suhara, Y., Doan, A., and Tan, W. (2020). Deep Entity Matching with Pre-Trained Language Models. *Proceedings of the VLDB Endowment*, 14(1):50–60.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692.

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., and Raghavendra, V. (2018). Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the SIGMOD Conference*, pages 19–34. ACM.

Newcombe, H., Kennedy, J., Axford, S., and James, A. (1959). Automatic Linkage of Vital Records. *Science*, 130(3381):954–959.

Shen, W., Wang, J., and Han, J. (2015). Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460.

Stonebraker, M. and Ilyas, I. F. (2018). Data Integration: The Current Status and the Way Forward. *IEEE Data Engineering Bulletin*, 41(2):3–9.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is All you Need. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 5998–6008.