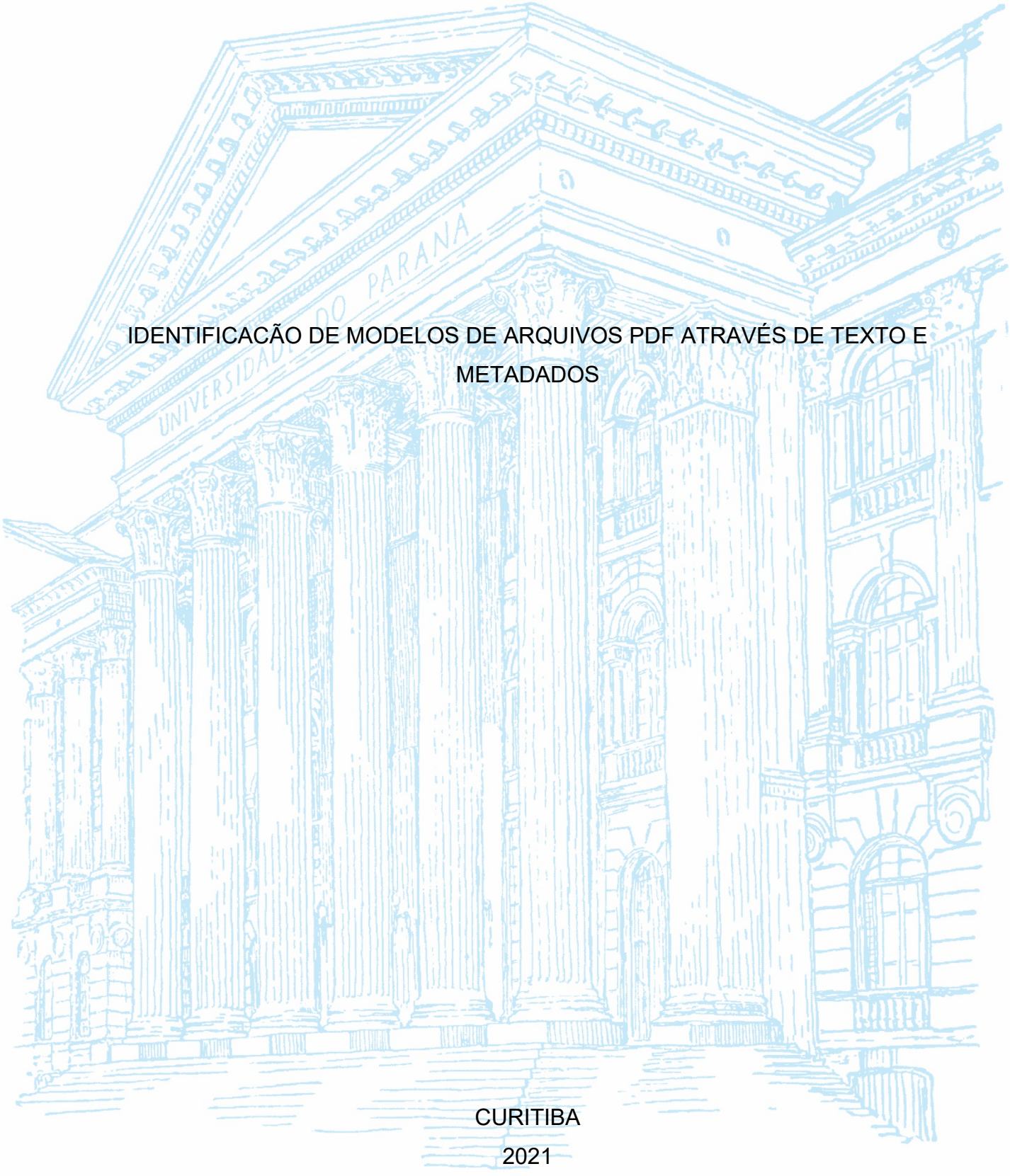
UNIVERSIDADE FEDERAL DO PARANÁ

MOZART HASSE



IDENTIFICACÃO DE MODELOS DE ARQUIVOS PDF ATRAVÉS DE TEXTO E METADADOS

CURITIBA

2021

Mozart Hasse

IDENTIFICACÃO DE MODELOS DE ARQUIVOS PDF ATRAVÉS DE TEXTO E METADADOS

Trabalho de conclusão de curso apresentado ao curso de Especialização em Inteligência Artificial Aplicada, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Especialista em Inteligência Artificial Aplicada.

Orientador: Prof. Dr. Jaime Wojciechowski

CURITIBA

2021

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INTELIGÊNCIA ARTIFICIAL APLICADA da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **MOZART HASSE** intitulada: **Identificação de modelos de arquivos PDF através de texto e metadados**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua ___APROVAÇÃO___ no rito de defesa.
A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 25 de Novembro de 2021.

JAIME WOJCIECHOWSKI
Presidente da Banca Examinadora

RAZER ANTHOM NIZER ROJAS MONTAÑO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Identificação de modelos de arquivos PDF através de texto e metadados

Mozart Hasse

UFPR - Universidade Federal do Paraná
Curitiba, Brasil
hasse@ufpr.br

Jaime Wojciechowski
Setor de Educação Profissional e Tecnológica
UFPR - Universidade Federal do Paraná
Curitiba, Brasil
jaimewo@ufpr.br

***Resumo*—O grande volume de documentos PDF gerados por aplicações comerciais exige o uso de algoritmos de classificação eficientes para organizar estes arquivos de maneira automática. Este trabalho apresenta uma solução para o caso específico em que seja necessário diferenciar modelos de documentos sujeitos a alterações constantes na organização dos dados a fim de viabilizar um tratamento posterior bastante específico para cada classe. O algoritmo escolhido é o KNN com similaridade de documentos usando o texto junto com os metadados e medida via *shingling*. Algumas opções de algoritmos são analisadas e comparadas com a implementação proposta. Os resultados são comparados numa base de referência e então conclui-se que a abordagem escolhida atende aos requisitos propostos e é satisfatória para esta aplicação.**

Palavras-chave*—classificadores, PDF, metadados, *shingling

***Abstract*—The large volume of PDF documents generated by commercial applications requires the use of efficient classification algorithms to organize these files automatically. This work presents a solution for the specific case in which it is necessary to differentiate document models subject to constant changes in data layout to enable a subsequent treatment that is very specific for each class. The chosen algorithm is KNN with document similarity using text and metadata and measured via *shingling*. Some algorithm options are analyzed and compared with the proposed implementation. The results are compared on a reference basis. The results show that the chosen approach meets the proposed requirements and is satisfactory for this application.**

Index Terms*—classifiers, PDF, metadata, *shingling

I. Introdução

Aplicações comerciais geram uma quantidade e variedade cada vez maior de documentos com finalidades e conteúdos diversos [1]. É bastante comum fazer com que estes dados sejam acessíveis ao usuário de forma razoavelmente padronizada usando arquivos PDF. Arquivos PDF possuem a vantagem de preservar não só o conteúdo como também a aparência ao longo do tempo, independente de plataforma [2].

Estudos como o citado por [3] indicam que 85% de todos os dados corporativos são capturados e armazenados em formato não estruturado. Estes arquivos podem vir de diversas fontes, incluindo aplicações comerciais de diversos fornecedores. Segundo [4], extrair dados confiáveis de documentos em formato PDF de fontes distintas apresenta diversas dificuldades devido a forma específica como a aplicação usada gera os dados de formatação.

Apesar do padrão PDF propor locais para colocação de metadados como autor ou título, estas informações raramente são colocadas dentro do documento com a precisão necessária para a correta identificação do conteúdo. As ferramentas que geram arquivos PDF priorizam a exibição, omitindo muitas informações ou dependendo do usuário para preenchê-las. E os usuários normalmente não dedicam tempo a esta atividade, buscando sempre o menor esforço possível [5].

Em algumas aplicações como aquelas que coletam metadados ou informações complementares sobre o documento, extrair todo o conteúdo não é essencial. Pode-se construir diversas aplicações usando apenas alguns dados do documento [6]. Basta ao computador identificar com precisão qual é o tipo de documento para saber onde está a informação de interesse se ela existir.

Um caso específico observado em aplicações comerciais é que, além do volume e variedade de conteúdo, existe ainda o problema de variação no formato. Aplicações que geram documentos PDF podem a qualquer momento lançar novas versões destes documentos. Alterações costumam incluir informações que antes não existiam ou trocar fontes, locais e tamanhos para organizar o conteúdo de outra forma.

Identificar onde exatamente está a informação desejada para extração e tratamento torna-se um problema cada vez mais complexo com o aumento do volume e variedade de formatos de documentos disponíveis. O problema pode ser ilustrado segundo a definição da IBM [7]:

- Volumes maiores de documentos exigem processamento automatizado;
- Velocidade de processamento passa a ser um problema se o gasto com processamento e separação de documentos crescer rapidamente. Se este gasto não for bem administrado, pode impedir a coleta da informação em tempo hábil para a tomada de decisão ou tornar a coleta inviável financeiramente;
- Variedade no formato dos dados pode exigir tempo e processamento adicional identificando como obter a informação desejada.

Processos de extração automáticos precisam saber qual é exatamente o formato e versão do documento para conseguir

buscar um dado específico de forma confiável. Desta forma, é possível seguir a tendência do mercado de automatizar essas tarefas, tornando os dados destes arquivos utilizáveis de forma ágil para a tomada de decisões gerenciais [1].

Um cuidado importante neste tipo de aplicação é que a proporção de tipos de documentos é bastante variada; a maioria dos documentos tem incidência bem pouco frequente, mas cada classe de documento precisa ser diferenciada dos demais. As restrições da Lei Geral de Proteção de Dados [9] tornam muito difícil e trabalhoso obter e manter quantidades significativas de documentos de cada tipo. Devido a isto, a base de documentos de treinamento possui um volume desproporcional de documentos de cada classe [10]. Nem todos os algoritmos lidam da mesma forma com essa variação, o que torna a complexidade da implementação bastante variada.

Via de regra a variedade de formatos e estilos dificulta a extração padronizada do texto. Por outro lado, pode-se conceber uma solução que se aproveite desta variedade para identificar exatamente o tipo e o modelo de documento. Propõe-se neste trabalho que é possível identificar com precisão o tipo de documento e a versão exata usada para sua construção simplesmente analisando seus metadados e um trecho do texto extraído.

Problemas de categorização de documentos são normalmente resolvidos de forma bastante eficiente por algoritmos de aprendizado de máquina. Há uma grande variedade de algoritmos, cada qual com suas aplicações, vantagens e desvantagens [6]. Este trabalho escolhe trabalhar especificamente no contexto de classificadores supervisionados [11], os seja, onde se possui um conjunto de documentos com classificações conhecidas e pretende-se usar a máquina para classificar automaticamente novos documentos dentre estas categorias.

Este artigo apresenta a implementação de uma solução para o problema de classificação de documentos, formulado da seguinte forma: dado um conjunto com grande variedade de documentos em formato PDF e suas respectivas categorias, usar algoritmos de aprendizado de máquina para construir um classificador. Este classificador será capaz de diferenciar tipos e variações de documentos a ponto de conseguir determinar - com precisão suficiente para extração confiável - quais são e onde estão as informações relevantes em cada documento. Deve-se assumir que a distribuição das classes é desbalanceada e sujeita a variações frequentes.

O restante deste artigo será organizado da seguinte forma: a Seção II detalha o problema e analisa os algoritmos e aplicações mais recentes na área de classificação de documentos. A Seção III descreve os critérios adotados, bases de dados e métodos de medição. A Seção IV descreve os procedimentos realizados e resultados obtidos. Por fim a Seção V analisa e avalia estes resultados para determinar se a abordagem utilizada serve como solução satisfatória para o problema proposto.

II. Fundamentação teórica

Um exemplo típico do problema de classificação de documentos abordado neste artigo pode ser entendido observando-se os trechos de documentos mostrados nas Figuras 1, 2 e 3. Apesar de todos os documentos serem artigos científicos com as mesmas informações básicas, alguns problemas ficam evidentes para a aplicação proposta:

- cada um dos editores coloca as mesmas informações em locais completamente diferentes dentro do documento, sem nenhuma marcação ou texto indicativo que permita ao computador determinar onde a informação fica;
- a orientação e o tamanho da letra mudam drasticamente (por exemplo no caso da data de publicação), assim, como o nível de detalhamento e formato da informação (por extenso, com ou sem o dia, etc.);
- mesmo sabendo-se o local aproximado da informação desejada, é preciso saber o que pode estar antes ou depois para conseguir filtrar corretamente o dado sem trazer junto nenhum outro valor indesejado. Um exemplo é dos e-mails no documento da Figura 1. Os e-mails no documento da figura estão agrupados por instituição e na mesma ordem dos autores (não agrupados) algumas linhas acima.

Um cenário concreto e específico para a ilustração do problema proposto é da Segfy [14], uma empresa de tecnologia que vende sistemas de gestão para corretores de seguros. Uma das principais funções do sistema é importar dados de apólices e propostas de seguros (ou seus endossos) para poder controlar comissões e pagamentos, fazer a segmentação da base de clientes e quaisquer outros controles relevantes para uma corretora de seguros. Neste ambiente há um grande volume e variedade de documentos diferentes entre as seguradoras e mesmo dentro da mesma companhia. O requisito básico é que o corretor envie os PDFs emitidos pela companhia para a aplicação, que determina automaticamente o tipo de cada documento e realiza a extração dos dados pertinentes de acordo com o tipo de seguro.

Os documentos precisam ser diferenciados de maneira bastante precisa pelos seguintes motivos:

- os termos técnicos usados variam de acordo com o tipo de documento. Por exemplo, o segurado é identificado como ”proponente”em uma proposta, mas como ”segurado”em uma apólice ou endosso. É necessário também identificar se o documento é de um tipo que permita extração para o sistema, pois vários documentos relacionados ao seguro (por exemplo a carta de comissão ou boletos e faturas) nem sempre podem ser importados para o sistema porque não possuem informações críticas para o cadastro;
- cada ramo de seguro tem termos e campos próprios. Seguros de bens (automóvel, casa, etc.) possuem dados diferentes do que seguros de Vida (beneficiários, benefícios, índices de atualização, etc.). Saber o ramo de que trata o documento é crítico para identificar quais campos podem ou não estar presentes;
- cada companhia usa termos próprios, mais informais (”preço do seguro”) ou mais jurídicos (”valor total do prêmio”), na redação dos documentos. É preciso saber quais termos procurar e quais seções estão antes e depois

do dado desejado para conseguir extrair a informação relevante com segurança;
- a ordem e nível de detalhamento dos dados muda conforme a seguradora, ramo e tipo de documento. É precisa saber quais informações estão presentes em cada documento para não correr o risco de procurar por uma coisa e confundir ou misturar com outra completamente diferente. Isto é particularmente importante neste ramo porque os documentos frequentemente possuem os dados organizados em 2 ou 3 colunas com tamanhos bastante variados;
- mesmo para a mesma seguradora, ramo e bem segurado, a todo o momento surgem novos modelos de documentos oferecendo produtos ou serviços personalizados. Pode ocorrer também que a companhia resolva reorganizar as informações para fidelizar mais determinado segmento de clientes ou manter um visual mais coerentes entre diversos segmentos de mercado;
- a ordem do texto dentro do documento muda conforme a ferramenta e a equipe que criou o documento. Em vários casos o valor do campo fica antes do título correspondente por questões de alinhamento, tamanho da fonte ou ferramenta usada.

Categorização de texto possui diversas abordagens e variações [13]. Uma solução de classificação de documentos pode ser analisada pelas seguintes dimensões:

- Aplicação da classificação;
- Representação dos dados utilizados para classificação;
- Algoritmos de classificação de documentos;
- Algoritmos para medição de similaridade de documentos;
- Soluções de mercado similares.

Cada caso é analisado separadamente a seguir.

A. *Aplicação da classificação*

Uma aplicação básica de um classificador aplicado a documentos é separar os documentos segundo o tipo de conteúdo [15]. Em [13] tem-se a descrição de inúmeras aplicações de processamento de linguagem natural para os mais diversos idiomas.

Alguns classificadores buscam apenas identificar a categoria de conteúdo em cada área da página, e no caso de [16] os tipos de documentos são limitados a artigos acadêmicos. Esse tipo de aplicação é denominada análise de leiaute de documento e é considerada uma etapa anterior a aplicações mais elaboradas [17].

Outras aplicações fazem a análise de leiaute com o objetivo específico de delimitar uma região de interesse para então extrair seu conteúdo, que pode ser um trecho de texto [18] ou um dado específico como o título [19].

Este artigo explorou o caso específico em que o objetivo é identificar o tipo de documento PDF a ponto de saber:

- os nomes dos campos presentes no documento;

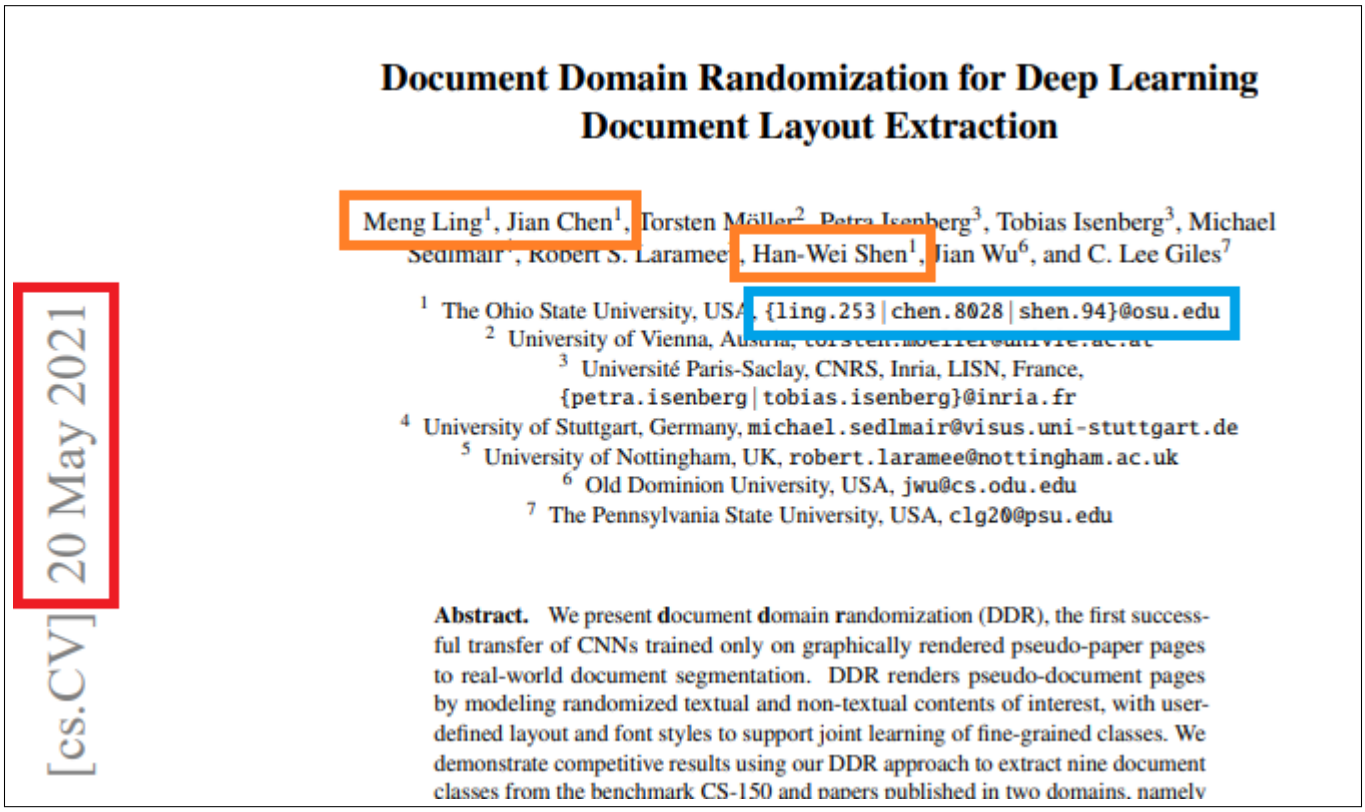


Figura 1. Exemplo de campos coletados a partir de um trecho de página de típico arquivo PDF. Campos relevantes como a data de publicação, nomes dos autores e e-mails correspondentes estão destacados para demonstrar sua típica localização em publicações desta editora. Fonte: [8]

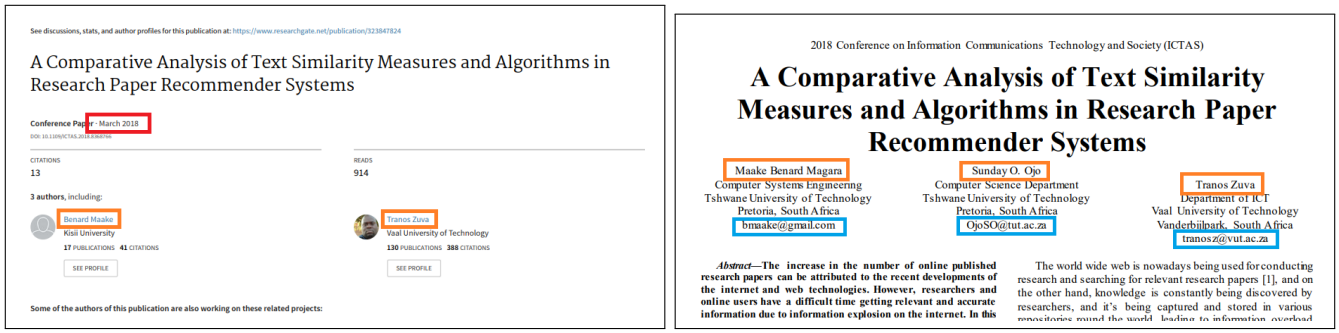


Figura 2. Exemplo de campos coletados a partir de trechos de páginas distintas de um típico arquivo PDF. Os mesmos campos estão destacados para comparação com a figura anterior. Neste caso a informação da data de publicação está incompleta e numa página diferente dos e-mails e nomes de todos os autores. Fonte: [12]

- posição aproximada de cada campo dentro de cada página, se o campo existir;
- nomes e localização aproximada dos campos próximos, acima ou abaixo da informação de interesse.

No caso específico deste artigo, o objetivo é somente determinar a classe dentre algumas dezenas de alternativas cuja localização dos campos é conhecida com alto nível de detalhamento. Para cada classe monta-se manualmente em uma etapa subsequente a lista de campos desejados e suas localizações aproximadas para a criação de uma lógica automatizada de extração.

B. Representação dos dados utilizados para classificação

Boa parte das implementações baseia-se apenas na imagem da página, como as aplicações de análise de leiaute [17] e muitas aplicações comerciais de extração de PDFs [20] [21] [22].

Outras abordagens similares analisam exclusivamente o texto do documento, como as que lidam com o processamento de linguagem natural [13] [23] [24]. Neste caso costumam-se analisar as palavras por ocorrência e relevância. Neste contexto é desejável reduzir o texto a suas palavras mais relevantes para reduzir a dimensionalidade e tornar o problema tratável computacionalmente. Abordagens deste tipo são tipicamente colocadas numa categoria chamada de saco de palavras (*bag-of-words*) [25]. Nessa perspectiva, costuma-se separar o pro-

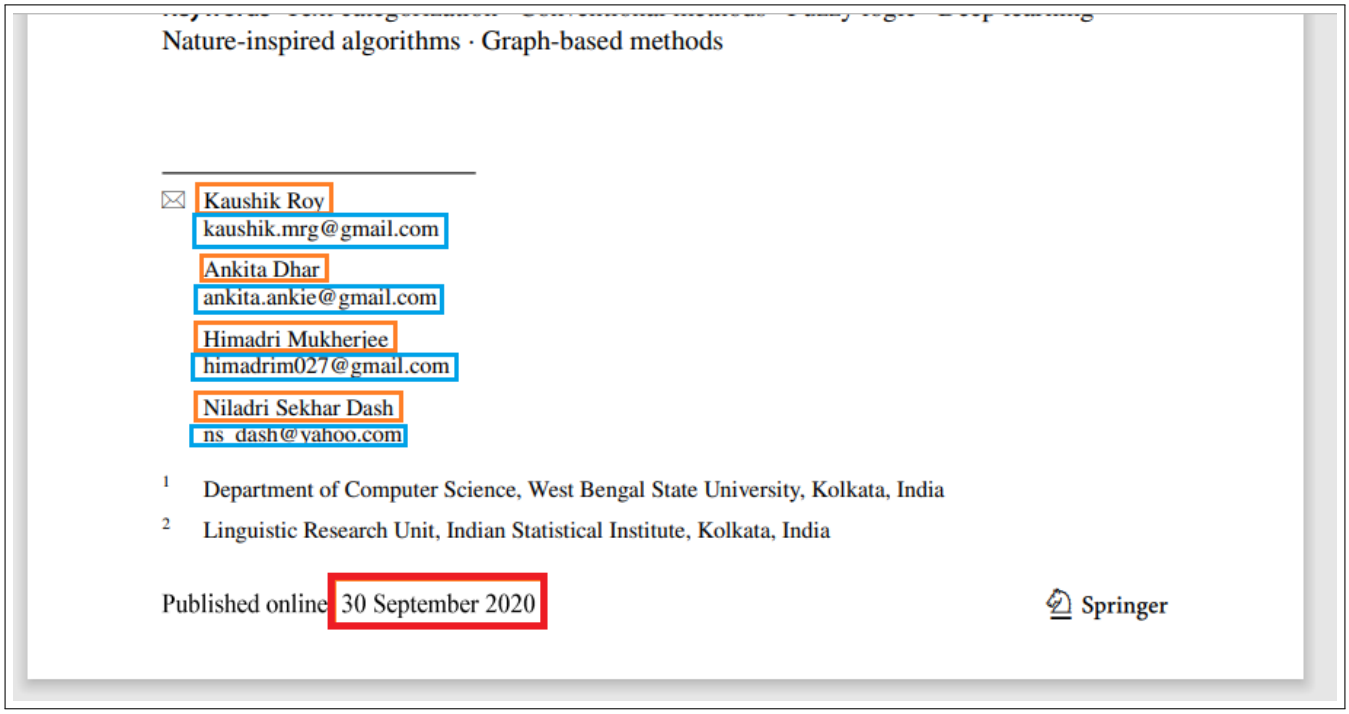


Figura 3. Exemplo de campos coletados a partir de trechos de páginas distintas de um típico arquivo PDF. Os mesmos campos estão destacados para comparação com as figuras anteriores. Neste caso nomes e e-mails estão no rodapé da página ao invés de estarem próximos ao título, além de não estarem agrupados por instituição. Fonte: [13]

cessamento do texto em duas etapas [15]:

- extração de características (*feature extraction*): delimita estruturas de linguagem e elimina o máximo possível de variações dependentes do idioma. Atividades típicas desta etapa são tokenização, remoção de palavras frequentes (*stop words*) e remoção de variações nas palavras (*stemming*);
- seleção de características (*feature selection*): extrai um subconjunto das palavras do texto de forma a preservar o máximo possível do conteúdo relevante para a aplicação. No caso de um classicador, o objetivo nesta etapa é escolher as palavras cuja frequência mais ajuda a diferenciar entre as classes de documentos. Várias abordagens são citadas em [15].

Um caso específico digno de nota é o de redes neurais baseadas em vetores de palavras [26]. O texto passa por um preprocessamento (tokenização, remoção de stop words, etc.) e depois as palavras são transformadas em um vetor de características que agrupa as palavras por algum critério específico. A frequência das ocorrências e co-ocorrências de cada palavras é usada como característica relevante para a classificação. O critério pode variar na quantidade de palavras e proximidade entre elas, ou mesmo ocorrência e co-ocorrência dentro do documento inteiro.

Alguns trabalhos tentam analisar o texto junto com as informações de formatação e localização [16] [18] [19]. Entretanto, o objetivo neles não é propriamente classificar o documento, apenas identificar regiões ou trechos específicos para extração de dados.

C. Algoritmos de classificação de documentos

Os mecanismos mais simples são apenas heurísticas baseadas em regras arbitrárias, como em [16]. Já no caso de [18] há também a tokenização e inclusão de estatísticas sobre as palavras na heurística usada, assim como o uso da imagem da página para determinar os limites de textos em colunas e cabeçalhos.

Dentre as alternativas que avaliam apenas o texto do documento, uma abordagem bastante comum é a de mudar a representação do documento, padronizando e filtrando apenas as palavras relevantes [15].

De acordo com [27] as principais categorias de algoritmos supervisionados para classificação de texto são:

- Algoritmo Rocchio [28]: este algoritmo classifica documentos através de sua representação como coordenadas num espaço vetorial. O treinamento do modelo consiste em determinar o centroide das instâncias de cada classe. Na hora da classificação o algoritmo atribui a uma nova instância à classe do centroide mais próximo. O que pode variar neste algoritmo são os critérios para medir distâncias nessa representação;
- K-vizinhos mais próximos (*k-Nearest Neighbor* - KNN) [28]: este algoritmo é baseado nas instâncias de treinamento, que são incorporadas ao modelo para classificação. Na hora de classificar uma nova instância, o algoritmo compara as distâncias de todos os exemplos conhecidos, e atribui a classe mais frequente dentre as classes dos K documentos mais próximos da instância a classificar. O que pode variar bastante neste algoritmo é o valor de K e o critério para medir a distância entre os documentos;
- Naïve Bayes [29]: os documentos são classificados em cada classe conforme as probabilidades de cada palavra ou elemento presente no texto. Essa probabilidade é calculada determinando-se, para cada elemento possível (seja ele uma palavra ou token), qual a probabilidade de um documento ser de determinada classe dado que aquele elemento está presente. Tendo-se as probabilidades de todos os elementos conhecidos, ao se classificar novos documentos basta calcular as probabilidades de cada um do elementos para cada classe possível e escolher a classe mais provável;
- Máquinas de suporte vetorial (*Support Vector Machines - SVM*) [30]: neste algoritmo o objetivo é achar, também num espaço vetorial, um conjunto de vetores que delimite a fronteira entre as classes. Além disso, estes vetores devem manter a maior distância possível entre as instâncias de quaisquer classes, além de ter igual distância entre as instâncias de cada classe. Estas características conferem a este algoritmo uma boa capacidade de generalização e um modelo bastante compacto por precisar apenas das instâncias necessárias para montar os vetores que separam as classes, chamados de vetores de suporte;
- Regressão [31]: os modelos de regressão operam também no espaço vetorial, de maneira similar ao SVM. No entanto, modelos de regressão buscam minimizar uma métrica de erro dada pela distância entre a classe dada para a instância de treino e a classe correta. Outra característica dos modelos de regressão é a construção de modelos baseados exclusivamente em simples funções lineares. Apesar de simples, segundo os autores os modelos de regressão conseguem resultados satisfatórios em tarefas de classificação de textos;
- Redes neurais: o caso mais comum de aplicação de redes neurais é baseado em texto, usando sacos de palavras ou vetores de palavras [26]. Neste caso a representação das palavras já está como enormes vetores ou matrizes de números que representam o conteúdo do texto. Basta treinar a rede neural fornecendo estes vetores e a classe esperada, como por exemplo em [32]. A topologia da rede pode variar, usando por exemplo redes de convolução [33]. Há casos em que se usam as imagens também, como em [34]. Uma visão bastante completa das alternativas mais recentes pode ser encontrada em [10].

Dentre soluções tão diversas, o algoritmo escolhido é detalhado em mais adiante. Antes disso, independente do algoritmo escolhido, é preciso definir como os documentos serão comparados uns com os outros.

D. Algoritmos para medição de similaridade de documentos

Conforme [35] e [12], pode-se agrupar os métodos de comparação de textos de documentos pelo seu grau de abstração, conforme mostrado a seguir:

- Similaridade de texto: esta é a abordagem mais simples e difundida. Cada documento é visto como uma sequência de caracteres ou palavras (*tokens*) e a similaridade é medida pela quantidade de trechos em comum entre os textos. As medidas se diferenciam pelo peso das diferenças e pela forma como tenta-se conciliar os trechos em comum de cada texto;
- Similaridade de contexto baseada em *corpus*: os documentos são reduzidos a conceitos ou palavras-chave e a similaridade entre eles é medida pela comparação de contexto com enormes bases de documentos. Assume-se que termos são similares se forem usados simultaneamente ou de maneira equivalente em muitas frases com palavras similares. Esta abordagem permite deduzir sinônimos ou termos correlatos a partir de grandes volumes de textos, porém o problema reside exatamente em montar uma base de textos grande o suficiente e corretamente organizada para permitir este aprendizado;
- Similaridade baseada em conhecimento (semântica): neste caso os documentos são comparados com base nos conceitos comuns que compartilham. Estes conceitos costumam ser representados diretamente através de múltiplos relacionamentos. Analogamente ao caso anterior, o desafio é organizar e manter atualizada essa lista de conceitos junto com seus graus de proximidade;
- Abordagens híbridas: conforme [35] há ainda diversas abordagens que não se encaixam adequadamente em nenhuma das anteriores ou que unem características de algumas delas para obter melhores resultados em contextos específicos.

Qualquer abordagem mais complexa do que a similaridade de texto recairia no seguintes problemas e limitações:

- exigiria uma base complementar personalizada para o contexto e tipo de documento a classificar. Isto tornaria a solução muito específica, assim como as conclusões que se poderia tirar de qualquer comparação entre algoritmos;
- os termos usados entre diferentes versões de documentos seriam potencialmente os mesmos ou extremamente parecidos, especialmente entre documentos da mesma empresa ou do mesmo ramo de atuação. Achar diferenças entre versões destes documentos num contexto como este dependeria de detalhes extremamente sutis e que dificilmente poderiam ser generalizados para situações além da base usada para treinamento;
- os metadados não seriam usados de maneira eficiente para diferenciar versões de documentos, pois um modelo baseado na semântica dos termos usados nos metadados não teria como generalizar sem um grande volume de exemplos;
- no contexto do problema proposto (diferenciar modelos de documentos de seguradoras), onde a variedade de documentos e versões é enorme, não há como esperar grande volume de documentos de cada tipo para alimentar o modelo antes dele ser colocado em prática;
- A quantidade desbalanceada de documentos de cada classe limita a capacidade do modelo em generalizar, especialmente se for baseado em palavras ou expressões.

A quantidade de trabalhos que usam uma abordagem de classificação de documentos similar é limitada porque a maioria dos classificadores se preocupa em classificar os textos pelo assunto e não pelo formato específico do documento. Devido a isso, a maioria dos classificadores de documentos se baseiam predominantemente em vetores de palavras ou outras abordagens baseadas em conhecimento semântico. Por outro lado, o foco deste trabalho é no uso de métricas de similaridade baseadas em caracteres ou em estatísticas sobre a ocorrência de palavras, restando neste contexto os algoritmos KNN e Naïve Bayes. Entre estes dois casos, [36] indica vantagem para o algoritmo KNN num contexto similar ao do problema proposto neste artigo.

Sobre o algoritmo KNN ainda resta escolher uma métrica eficiente para medir a similaridade ou a diferença entre documentos. Um caso bastante relevante é o *locality-sensitive hashing* (LSH) [37]. O objetivo desta técnica é encontrar dados similares em grandes bancos de dados, abrindo mão da precisão absoluta porém com um enorme ganho de desempenho e uma margem de erro configurável. Em essência esta técnica gera um *hash* para cada documento a comparar, sendo que este *hash* tem a característica especial de que é calculado de forma que documentos similares tenham *hashes* com uma proporção parecida de similaridade. Uma vez criado o *hash* para cada documento, basta comparar os *hashes* ao invés dos documentos. Números de *hash* próximos indicarão documentos ”próximos”(parecidos) com alta probabilidade. Existe sempre uma probabilidade de erro, porém ela pode ser ajustada pelo tamanho do *hash* [37].

Sobre essas métricas de similaridade, em [38] há bastante material recente sobre técnicas baseadas em *locality-sensitive hashing*, indicadas especialmente para casos em que o desempenho é crítico devido ao volume de documentos.

E. Soluções de mercado similares

Algumas soluções de mercado encontradas para classificação de documentos procuram gerar uma classificação muito mais relacionada ao tema do documento, não necessariamente seu formato, como por exemplo [23], que classifica o documento lido a partir da imagem renderizada. O caso de [24] é similar, porém exige um número considerável de exemplos para cada classe.

Outro caso é o de [39], que usa o texto dos documentos, porém exige um número mínimo de exemplos, contraexemplos e textos esperados para cada classe. Outro limitador é que essa solução só gera classificadores binários (ou seja, o documento é ou não é de determinada classe). Para casos em que há uma enorme variedade de classes possíveis, gerar e manter estes classificadores seria bastante trabalhoso e ineficiente.

As soluções encontradas em sistemas de gestão similares ao da Segfy são predominantemente baseadas na extração de texto através de bibliotecas de leitura de PDF de código aberto, com o tratamento do texto extraído através filtros de expressões regulares. Esta abordagem apresenta diversos problemas e potenciais limitações, devido a nuances na construção dos documentos PDF e variações nas diversas versões destes documentos.

Diferenciar versões do mesmo tipo de documento de uma mesma companhia é uma tarefa praticamente impossível observando-se apenas o conteúdo de texto. A similaridade é muito grande entre as diversas versões, frequentemente variando-se apenas a ordem de alguns itens e as posições de espaçamentos e quebras de linhas. O problema é que estas pequenas variações no texto são exatamente as coisas que se pretende identificar antes de tentar a extração, portanto limitando bastante a quantidade e a qualidade dos dados que de pode extrair de cada documento. Abordagens baseadas em imagem também não se mostram viáveis, pois o tempo de conversão do PDF em imagem e sua classificação se mostram proibitivos para o volumne de documentos processados neste tipo de solução.

III. Materiais e métodos

As etapas necessárias para implementar a solução descrita neste artigo foram as seguintes:

- Escolha da base;
- Preparação dos dados;
- Cuidados com a generalização e dados pessoais;
- Escolha e aplicação dos algoritmos;
- Configuração dos parâmetros;
- Arquitetura da implementação;
- Critérios para análise da eficácia do algoritmo.

Cada uma das etapas é detalhada a seguir.

A. Escolha da base

Considerando que todos os elementos a classificar já foram definidos como sendo arquivos em formato PDF, basta confirmar que cada elemento tenha uma classe atribuída e então definir por qual meio os dados relevantes são extraídos. Especificamente no caso da empresa em questão, já existia uma base histórica de solicitações de clientes com cerca de 4226 arquivos PDF, com cada seguradora em uma pasta diferente e a distribuição de classes indicada na Tabela I.

Se esta base não existisse ou já não estivesse classificada, uma opção seria gerar documentos aleatórios, pois a obtenção de arquivos para montar um classificador seria burocrática e demorada. Gerar documentos aleatórios como em [8] é muito conveniente quando há a necessidade de um volume enorme de dados similares para o treinamento. Este costuma ser o caso em classificadores baseados em imagem, como redes neurais convolucionais.

Apesar de possível e permitir a eliminação de dados pessoais, esta alternativa exigiria um esforço enorme criando formatos iguais aos dos documentos originais. Isto ocorre porque a quantidade e variedade de itens a combinar para formar uma página seria enorme enquanto que a diferença na localização dos elementos entre um formato e outro seria muito pequena.

Pasta(classe)	Documentos
digitalizado	1
seg1	3
seg2	1
seg3	56
seg5	256
seg6	11
seg7	13
seg8	30
seg9	398
seg10	10
seg11	18
seg12	224
seg13	4
seg14	31
seg15	66
seg16	4
seg17	28
seg18	95
seg19	11
seg20	302
seg21	31
seg22	1
seg23	36
seg24	26
seg25	478
seg26	461
seg28	39
seg29	510
seg30	4
seg31	8
seg32	81
seg33	3
seg34	221
seg35	55
seg36	245
seg37	16
seg38	276
seg39	29
seg40	28
seg42	116

Tabela I
Classes iniciais e quantidade de documentos usados para treinamento. Fonte: o autor

Tendo inicialmente apenas a capacidade de diferenciar companhias, o modelo pode ser aprimorado de maneira controlada e mensurável adicionando-se uma nova classe por vez. A fim de simplificar outros detalhes de implementação e integração preferiu-se diferenciar os tipos de documentos de uma companhia de cada vez, medindo-se os resultados após separar todos os documentos conhecidos de uma das companhias da base de treinamento.

B. Preparação dos dados

Conforme já observado em outros artigos [19] a preferência é pelo uso de ferramentas genéricas de código aberto como PDFBox [40].

De acordo com [30], frequentemente os maiores ganhos de performance num classificador são provenientes mais da exploração de características específicas do domínio do que da troca de um algoritmo por outro. Do mesmo modo, buscou-se extrair de cada documento apenas as informações necessárias

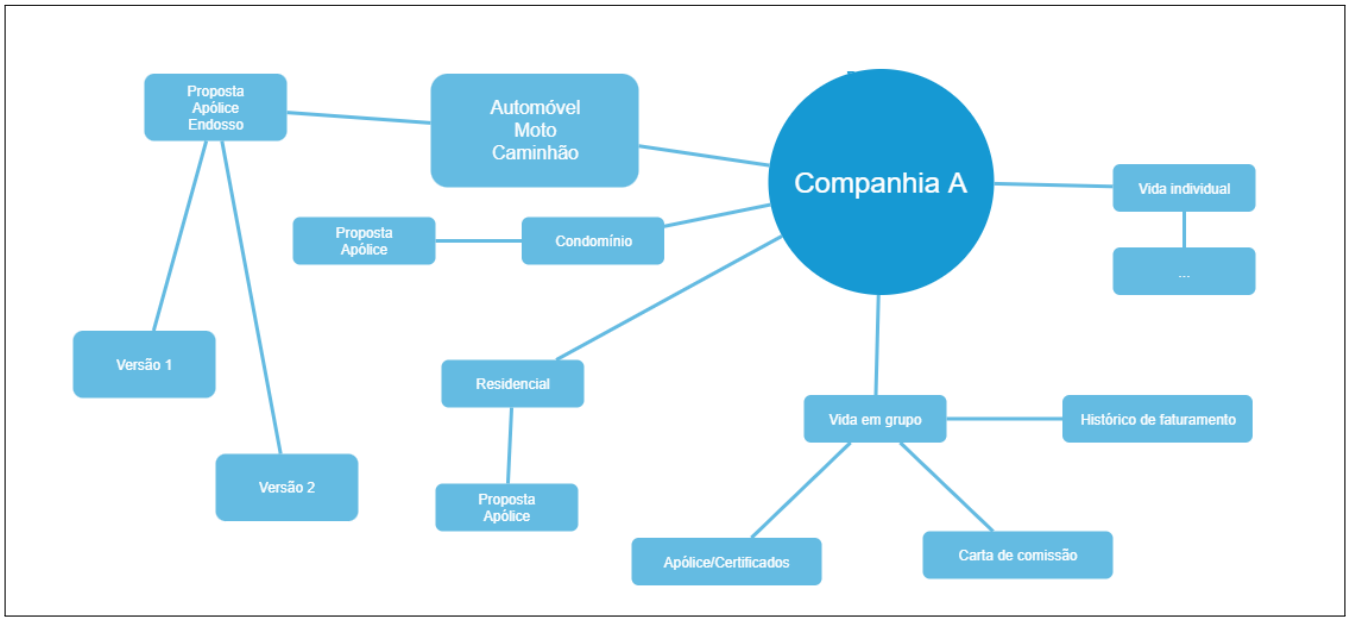


Figura 4. Exemplo de detalhamento de documentos para uma das companhias reconhecidas. Note-se o agrupamento dos tipos apólice, proposta e endosso devido ao fato de que para essa companhia esses documentos possuem o mesmo formato, com os mesmos campos praticamente na mesma posição e ordem. Fonte: o autor

e suficientes para diferenciar modelos e versões entre si. Os dados considerados foram:

- Metadados contidos no arquivo PDF, mesmo que incompletos ou não preenchidos;

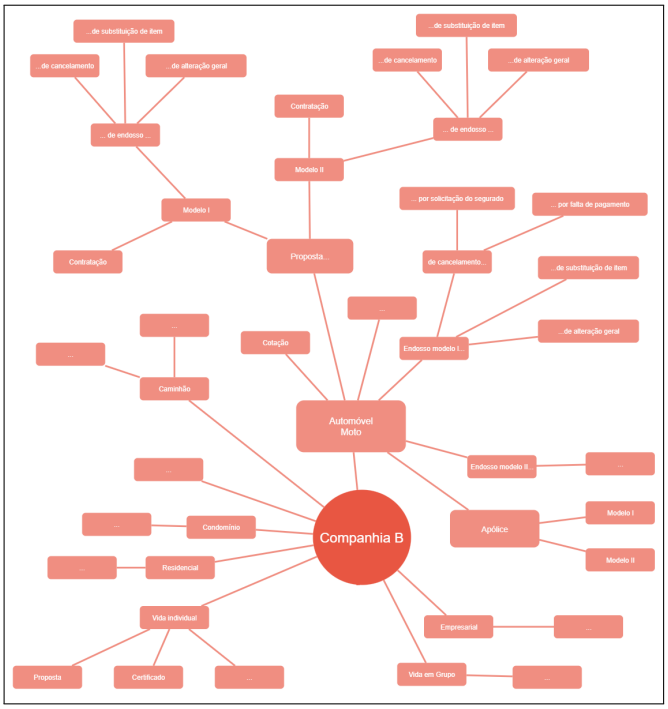


Figura 5. Exemplo de detalhamento para outra companhia. Note-se o agrupamento diferente dos documentos para cada ramo, além do detalhamento e subdivisão em modelos com leiautes diferentes, cada qual com campos em posições e ordem diferentes. Fonte: o autor

- Texto extraído de forma padronizada usando o mesmo mecanismo para todos os documentos. Uma forma simples de garantir a reproducibilidade foi usar como base o texto exatamente como extraído pelo PDFBox [40];
- Informações resumidas sobre objetos não-textuais, por ordem de aparição. No caso de imagens e demais campos com dados binários, apenas o tamanho e a ordem dentro da página foram considerados.

Apesar de perder alguma informação que evidencie a diferença entre uma versão de documento e outra, partiu-se da suposição de que estas informações seriam suficientes para diferenciar modelos e versões de documentos. Isto também torna conveniente a manipulação dos dados pelo classificador pela expressiva redução no volume total de bytes analisados por documento.

Em suma, cada documento é transformado em um pequeno texto com cerca de 6000 bytes que contém os campos de metadados com os respectivos valores, o texto inicial do documento e mais uma lista de metadados sobre as imagens e suas coordenadas. Pode-se considerar que este texto é um ”resumo”do documento e contém toda a informação necessária para definir qual a sua classe num contexto como o das Figuras 4 e 5.

Considerando o tipo de documento a classificar e o contexto, cabe notar que muitos campos dentro dos documentos são opcionais e vários deles raramente são preenchidos. Devido a isso, as quebras de linhas, espaços e tabulações repetidos no meio do texto foram removidos. O objetivo desta modificação é que o processo de classificação dê mais ênfase à informação efetivamente preenchida no documento, tanto de texto quanto metadados. O raciocício por trás desta decisão é que longas sequências de espaços e tabulações não contribuem para dife-

renciar um documento do outro e apenas ocupariam espaço. Isto é evidenciado pelos seguintes fatos:

- longas sequências de espaços são encontradas em grande quantidade em documentos de todas as classes;
- dentro de documentos da mesma classe, é possível encontrar sequências de espaçoes com tamanhos variados, causadas entre outros fatores pela presença de campos consecutivos opcionais e não preenchidos;
- como se verá mais adiante, os documentos serão representados e classificados baseando-se em *hashes* de pequenos trechos de texto. Neste contexto, *hashes* baseados em múltiplos espaços em branco ou quebras de linha acabariam ”poluindo”o *hash* dos documentos com informação potencialmente inútil, o que geraria falsos positivos e um número maior de documentos com o mesmo grau de semelhança.

A base inicial de documentos foi separada de maneira aleatória em uma parte de treinamento e outra de testes. O conjunto de documentos usados tinha quantidades muito variadas por classe, por isso o único requisito para divisão dos dados foi de que a base de treinamento tivesse ao menos um documento de cada classe. O objetivo inicial foi validar a capacidade de diferenciação da solução criando-se um classificador capaz de identificar apenas a companhia. Os valores dos principais parâmetros e os detalhes de implementação foram baseados nos resultados para este caso, medidos sobre a base de testes. Após a validação da solução, uma das companhias foi escolhida arbitrariamente para ter seus documentos divididos por tipo e os resultados foram acompanhados com dados reais de clientes por um período.

C. Cuidados com a generalização e dados pessoais

Considerando os cuidados recentes com a manipulação e retenção de dados pessoais [9], buscou-se evitar ao máximo o armazenamento ou manipulação de dados considerados sensíveis, tais como números de documentos, datas de nascimento e datas/horas de criação ou manipulação de arquivos. Pode-se refletir que isso é até benéfico para o processo de classificação, já que estes valores deveriam ser irrelevantes para determinar o tipo de documento.

Baseando-se nestas suposições, como preprocessamento substituíram-se todos os dígitos visíveis do texto extraídos dos documentos por um caractere neutro: ”_”. Além disso, foram eliminados trechos dos metadados que tinham referências a datas e horários, ou seja, textos em formatos como *xx/xx/xxxx* ou *xx:xx:xx*. Outros valores numéricos nos metadados como tamanhos, coordenadas e números de versões foram mantidos apenas nos metadados, pois supõe-se que ajudem a diferenciar entre versões de modelos de documentos.

O objetivo destas mudanças também é evitar que os documentos sejam comparados com base em documentos pessoais ou provavelmente não relacionados com o formato do documento tais como datas e horas. Espera-se também que ao deixar datas, horas e documentos como tokens padronizados a sua localização em relação a outras palavras seja suficiente para contribuir na identificação do tipo de documento e seu formato.

Ainda no contexto específico da base de dados usada, constatou-se que a quantidade de páginas varia muito entre os documentos, podendo em alguns casos chegar a quase 100. Entretanto, toda essa informação não é necessária para diferenciar o formato e os tipos de dados presentes no documento. Usando-se a constatação empírica de que um ser humano consegue identificar inequivocamente o modelo de documento (seguradora, tipo de documento e leiaute básico) olhando apenas o começo do documento, decidiu-se usar como entrada para o classificador apenas um trecho do texto no começo de cada documento, além dos metadados.

Buscando-se também eliminar a necessidade de incorporar e controlar dados pessoais dentro do modelo, pode-se eliminar manualmente a pouca informação pessoal restante, que seriam os nomes de pessoas citados no começo de cada documento. No caso cada documento incorporado ao modelo de treinamento é manualmente complementado com os nomes e sobrenomes a remover, quando aplicável. Os termos indicados são substituídos por termos neutros antes de serem usados para o treinamento. Desta forma, o modelo pode ficar completamente livre de dados pessoais identificáveis. Apesar de simples, este procedimento gera um trabalho considerável e não é necessário para medir a precisão de classificação do algoritmo. Portanto, o processo de medição de eficiência do algoritmo não levou em consideração a presença ou ausência de dados pessoais nos documentos analisados.

D. Escolha dos algoritmos

O algoritmo foi escolhido com base nos principais requisitos vistos como essenciais para viabilizar esta aplicação para um grande volume de documentos. O tempo de treinamento deve ser curto, pois o volume de documentos é muito grande e um tempo alto de treinamento deixaria a aplicação parada a cada mudança. A inclusão de novas classes deve ser rápida, pois deve-se assumir que novos modelos de documentos surgem a todo o momento, como evidenciado nas Figuras 4 e 5.

Usando-se o raciocínio por exclusão, considerou-se justificável descartar as redes neurais devido ao tempo de treinamento mais alto do que os demais algoritmos [10]. Apesar dos outros algoritmos apresentados terem um tempo de treinamento não muito grande, nenhum deles se compara ao tempo de treinamento do KNN, que é baseado em instâncias e por isso mesmo só precisa da inclusão de um exemplo da nova classe para ser capaz de classificá-la imediatamente com um razoável grau de precisão.

Mesmo que o tempo de classificação do KNN seja relativamente alto por causa da necessidade de comparar todos os elementos da base com a instância a classificar, este tempo não chega a ser um problema pelos seguintes motivos:

- o volume de dados comparado de cada documento é na maioria dos casos extremamente baixo por causa do uso de *hashes*, o que faz com que apenas hashes sejam comparados entre si;

- a quantidade de exemplos de documentos a manter na base de comparação é relativamente pequena. Conforme será mostrado durante a fase de experimentos, bastam poucos documentos para atingir um ótimo nível de classificação;
- apesar de frequente, a inclusão de novos tipos de documentos no classificador não implica na inclusão de grandes quantidades de instâncias;
- o tempo gasto em treinamento é crítico para este tipo de aplicação. O uso de KNN permite um ajuste nos critérios de maneira praticamente instantânea, bastando incluir na base qualquer documento incorretamente classificado. Outros algoritmos exigiriam algum tempo gasto numa etapa de treinamento, durante a qual o classificador continuaria operando sem reconhecer adequadamente o documento incorretamente classificado.

Note-se que o algoritmo KNN não resolve todos os problemas. De nada adianta uma complexidade baixa e proporcional ao número de instâncias do modelo se a comparação de distância entre os documentos não tiver um custo computacional igualmente baixo. A maioria dos algoritmos para comparar textos palavra por palavra tem um custo computacional relativamente alto, de ordem quadrática. Uma exceção bastante interessante por manter uma complexidade linear é da categoria de *locality-sensitive hashing*. A grande vantagem dessa categoria de algoritmos é que o cálculo do *hash* é bastante rápido e os *hashes* dos documentos podem ser usados diretamente para medir a similaridade com alta probabilidade [37].

Um excelente exemplo de *locality-sensitive hashing* é o uso de *shingles* [38], que possuem um tamanho configurável e, por serem simplesmente vetores de inteiros, podem ser comparados de maneira extremamente rápida com todo o modelo de classes conhecido. Considerando que o desempenho é uma prioridade nesse tipo de aplicação, resta concluir que esta opção é adequada para este caso por ser a maneira mais eficaz de representar documentos de maneira equivalente a conjuntos de palavras [38].

A implementação usou [41] como referência, porém com vários ajustes para se trabalhar com texto e metadados de arquivos PDF, além de simplificações e otimizações para a aplicação proposta. As definições usadas foram as seguintes:

- palavras frequentes (*stop words*) não precisaram ser removidas nem ter qualquer tratamento especial, pois segundo [38], quando consideradas em conjuntos de pelo menos 3 palavras, são capazes de formar *hashes* relevantes para auxiliar na identificação do documento;
- o mecanismo usado para gerar um resumo do documento que possa ser usado para comparação com outros documentos foi a técnica de *minhashing*. Conforme indicado por [38], a probabilidade de dois documentos terem o mesmo *minhash* é a mesma do que a similaridade de Jaccard entre estes mesmos documentos;
- as funções para geração dos *minhashes* têm o formato $f(x) = ax + b \bmod p$, onde a e b são números inteiros aleatórios de 64 bits, x é a representação numérica do grupo de palavras do documento (CRC-32 [42]) e p é um número primo fixo com 64 bits. O tamanho de 64 bits foi escolhido por ser adequado para processamento nativo de inteiros pelos processadores atuais, oferecendo bom desempenho.

A primeira definição necessária para o uso de *hashes* é de que forma os documentos devem ser divididos para calcular os *hashes*. No contexto da aplicação proposta, o ponto considerado necessário e suficiente para identificar e diferenciar os documentos foi o de pequenos trechos com poucas palavras. O motivo para essa escolha é que apenas uma palavra, olhada individualmente, não é suficiente para identificar o modelo ou versão de documento. Muito pelo contrário, palavras do jargão de seguros (por exemplo ”segurado”, ”coberturas”, ”prêmio”) são bastante comuns e estão presentes em proporção parecida em quase todos os documentos da área. Por outro lado, sequências com cerca de 3 ou mais palavras (por exemplo trechos de frases no meio do questionário de risco) são bastante informativas e podem ser determinantes para identificar uma companhia e um modelo de documento.

Durante a implementação, testes preliminares demonstraram que o classificador conseguia chegar em resultados minimamente satisfatórios a partir de 3 palavras dentro do *hash*. Colocar trechos maiores fez pouca diferença nos resultados, e intuitivamente frases longas demais podem acabar tornando os *hashes* excessivamente específicos por aumentar a chance de incluir algum dado variável do documento dentro do cálculo. Exemplos de dados indesejados no *hash* incluem informações sobre a localização (bairro, cidade ou estado do segurado, condutor ou corretora) ou do bem segurado (modelo do veículo ou características específicas).

Este tipo de informação específica de cada documento inevitavelmente contribui para a formação do *hash*, porém a intenção é minimizar o seu impacto na representação do documento. Se o modelo não tiver mecanismos de generalização como este, uma possibilidade bastante provável é a do classificador escolher a classe do documento mais com base nos dados do segurado (que são variáveis) do que nos dados esperados do modelo de documento (que são fixos). A maneira ideal de evitar problemas deste tipo é ter mais exemplos de documentos de forma que a parte fixa do documento esteja mais presente e seja mais frequente do que dados específicos e presentes em apenas um documento do conjunto de treinamento. Apesar de não se ter garantias já que *locality-sensitive hashing* é por definição um método randomizado [37], buscou-se com esta medida minimizar os potenciais problemas a nível de modelagem.

Apesar do uso de *locality-sensitive hashing* lidar muito bem com o problema de desempenho, um problema que ainda precisa de tratamento é o de como montar o critério de desempate para os casos em que um novo documento ao ser classificado ter a mesma quantidade de *hashes* comuns do que vários outros de várias classes. Este problema pode surgir com expressiva frequência por causa da natureza da base de dados: dezenas de classes diferentes, muitas delas

com poucos exemplos, em alguns casos apenas um. Usar números ímpares de vizinhos mais próximos (o K do algoritmo KNN) não ajuda muito neste caso, pois empates ainda podem acontecer. Algumas opções foram verificadas para resolver este problema.

Uma alternativa seria, ao invés de montar um só classificador para todas as classes, montar vários classificadores binários, um para cada classe, onde cada classificador indica se o documento pertence ou não à classe correspondente. Há algumas formas eficientes de organizar classificadores binários para diferenciar entre múltiplas classes [43] e com isso organizar e dividir o problema de classificação de documentos em vários problemas menores. Dentro de cada classificador binário não haveria como ter empates simplesmente usando-se um número impar para o parâmetro K do algoritmo KNN.

As abordagens de classificadores binários resolveriam apenas parcialmente este problema, pois ainda poderiam ocorrer classificações positivas do mesmo documento por classificadores binários diferentes. Isso seria descartado se os classificadores binários forem organizados em estruturas hierárquicas como árvores [43], porém a hierarquia precisaria ser refeita com a inclusão de novas classes, e o resultado do classificador seria influenciado pela escolha da hierarquia, que precisaria ser reorganizada para se chegar numa boa precisão. Toda essa complexidade exigiria também retreinamentos frequentes de todos os classificadores, o que tornaria o modelo grande e com custo computacional maior para se manter do que o esperado para um único classificador de múltiplas classes. Por esse motivo, adotou-se um único classificador para todas as classes enquanto que a solução para o problema dos empates ficou para ser resolvido de outra forma.

A solução natural é usar alguma métrica de proximidade para ver qual classe estaria mais próxima do exemplo a classificar. Entretanto, não faz sentido comparar distâncias entre *hashes* individuais, assim como não adianta elevar muito o número de *hashes* na comparação porque sempre poderão ocorrer empates. É preciso usar um outro algoritmo para medir com precisão a distância entre documentos nos raros casos em que o *locality-sensitive hashing* não dê uma única opção como resposta.

A solução escolhida para lidar com situações onde mais de um documento do modelo tenha exatamente a mesma quantidade de *shingles* em comum com o documento a classificar foi a seguinte:

- Passo 1: monta-se uma lista com as classes de todos os documentos com a mesma quantidade de *shingles* e suas classes. Computam-se então os totais de documentos para cada classe;
- Passo 2: se houver uma classe com mais documentos do que as demais ou se todos os documentos forem da mesma classe, esta classe é escolhida para classificar o novo documento;
- Passo 3: se duas ou mais classes tiverem exatamente a mesma quantidade de documentos com a mesma quantidade de *shingles*, então escolhe-se aleatoriamente um documento de cada classe e faz-se a comparação entre os textos do documentos usando-se uma função de distância [44]. Dentre os documentos candidatos escolhidos de cada classe, o que tiver a menor distância do documento a classificar indicará a classe correspondente.

Dentre as métricas de similaridade de texto possíveis, levaram-se em conta os seguintes fatores:

- o algoritmo teria de comparar textos não muito grandes (média perto de 6000 bytes), porém com tamanho máximo indeterminado, o que torna métodos como a distância de Hamming [44] não tão eficientes por ser necessário fixar arbitrariamente um valor máximo e truncar os dados que ultrapassarem este limite;
- considerando a importância de ter um bom desempenho, métricas que se baseiem na frequência relativa das palavras (*term frequency–inverse document frequency* ou TF-IDF) ou em vetores de palavras (distância por cosseno) [45] também foram descartadas;
- variações na ortografia e nos termos usados são relevantes para diferenciar versões de documentos, portanto métricas baseadas em semelhança sonora também devem ser descartadas;
- a frequência de uso deste algoritmo é relativamente baixa pois só é necessário caso haja empate na comparação dos *hashes*.

Considerando as condições acima, restam opções de medição de similaridade de edição. As principais métricas deste tipo são a similaridade de Jaro [46] e a similaridade de Jaccard [44]. A similaridade de Jaro é calculada pela seguinte fórmula:

$$f(d_1, d_2) = \begin{cases} 0, & \text{se } c \text{ é zero} \\ \frac{1}{3}\left(\frac{c}{|d_1|} + \frac{c}{|d_2|} + \frac{0.5t}{c}\right), & \text{caso contrário} \end{cases}$$

Onde:

- d_1 e d_2 são dos textos a comparar
- $|d_1|$ e $|d_2|$ indicam o comprimento de cada texto
- c é a quantidade de caracteres em comum entre os dois textos (apenas os caracteres que estiverem a menos de metade do tamanho do menor texto)
- t é a quantidade de transposições necessárias para deixar os caracteres fora de ordem nas mesmas posições

Já a similaridade de Jaccard é calculada pela seguinte fórmula:

$$f(d_1, d_2) = \frac{d_1 \cap d_2}{d_1 \cup d_2}$$

Onde o critério de similaridade pode ser o de caracteres em comum ou (em casos de textos maiores) palavras em comum entre os documentos.

Apesar de muito similar à de Jaro, a similaridade de Jaccard foi descartada por ser equivalente à comparação de *shingles*, não acrescentando portanto uma métrica relevante para desempate entre os documentos.

E. Configuração dos parâmetros

Os parâmetros relevantes que precisam ser ajustados para este caso são:

- quantidade de *shingles* por documento: a quantidade deve ser suficiente para que a probabilidade de haver empates entre documentos seja muito pequena, sem aumentar demais o custo computacional em calcular uma grande quantidade de *shingles*. Empiricamente o valor escolhido foi 20 *shingles* por documento. A eficácia deste valor será melhor ilustrada na seção de experimentos;
- quantos documentos considerar na hora de escolher a classe (o "K"do KNN): apesar de relevante na maioria das situações, variar este valor mostrou ter importância muito menor do que a quantidade de *shingles*. Na quase totalidade de casos testados, quando a quantidade de *shingles* era suficiente o valor de K incluía apenas os documentos da classe correta. Portanto, por questão de simplicidade e desempenho adotou-se K=1, ou seja, a classe é a mesma do documento mais similar do modelo. Isto permite também que seja necessário informar apenas um documento da classe para obter um classificador com bom nível de acurácia, apesar de mais documentos serem desejáveis para manter o nível de erros bem próximo de zero. Apesar do K ser igual a 1, ainda podem ser considerados mais documentos caso o número de *shingles* em comum seja exatamente o mesmo. Somente neste caso entram os critérios de desempate no número de documentos por classe e a aplicação da similaridade de Jaro como último recurso.

F. Arquitetura da implementação

A implementação da solução foi feita inteiramente numa arquitetura em nuvem, usando o provedor de *IaaS* (infraestrutura como serviço) AWS [47] [48]. Todo o processamento é feito usando o modelo de computação "sem servidor"(*serverless*) [49], integrado com o restante da aplicação via chamada de API.

G. Critérios para análise da eficácia do algoritmo

A comparação entre as alternativas de implementação mais relevantes usa uma métrica simples e bastante comum de comparação: acurácia de classificação nos casos de teste:

$$acurácia = \frac{|TP|}{|T|}$$

Onde:

- *TP* é o número de documentos classificados corretamente;
- T é o total de documentos analisados.

Apesar da métrica proposta ser extremamente simples, o principal critério usado para medir a eficácia da solução foi a proporção de documentos corretamente classificados. Isto foi considerado satisfatório neste caso porque, independente do tipo de erro (falso negativo ou falso positivo), o resultado para o cliente é exatamente o mesmo: o documento classificado incorretamente não é importado com sucesso.

A solução proposta será comparada com outros algoritmos, todos analisados sobre as mesmas bases de treino e teste. Considerando que o volume de exemplos é razoável, optou-se por separar os dados da base original em um conjunto de treinamento com 70% dos dados e outro de teste com o restante [43]. A separação foi aleatória, tendo apenas uma etapa de revisão para garantir que a base de treinamento tivesse ao menos um exemplo de cada classe. Mesmo havendo alternativas que garantissem uma distribuição balanceada entre as classes, esta alternativa não foi usada neste caso.

A velocidade de treinamento, teste ou classificação não foi considerada relevante neste caso pois o objetivo é medir principalmente a acurácia do modelo. Devido a outros requisitos considerados relevantes para esta aplicação, a solução proposta já foi escolhida com base em seu desempenho. Portanto, resta apenas verificar se a diferença entre a implementação atual não teve perda significativa de acurácia em relação a algum outro algoritmo, já que no quesito complexidade o algoritmo escolhido está entre os melhores.

Considerando o algoritmo escolhido, os seguintes pontos foram considerados relevantes para medir a eficácia da implementação:

- Seletividade dos *hashes*: mesmo que a acurácia do modelo pareça satisfatória levando-se em conta os dados disponíveis, é possível conferir com maior profundidade qual é a efetiva participação do conceito de *locality-sensitive hashing* sobre o resultado. Isto pode ser estimado comparando-se a quantidade de *hashes* em comum entre documentos de mesma classe e documentos de classes diferentes;
- Quantidade de *hashes* por documento: um dos parâmetros a determinar para o funcionamento do algoritmo é a quantidade de *hashes* a coletar de cada documento. Devido a isso é necessário fazer alguns testes com valores diferentes, comparando a acurácia, a fim de determinar um valor ótimo aproximado para esta base e qual o impacto de variações no valor deste parâmetro;
- Comparação com outro algoritmo (Naïve Bayes): uma comparação exata de desempenho exigiria adaptar a base às características específicas e limitações do algoritmo Naïve Bayes. Idealmente as classes teriam de ter quantidades balanceadas e a tokenização teria de ter muito mais etapas para que as sequências de palavras pudessem representar de forma mais eficaz a classe de cada documento. Conforme explicado nas seções anteriores, fazer tudo isto neste contexto tornaria a aplicação praticamente inviável devido ao esforço necessário no preprocessamento dos dados;
- Medição de desempenho: o tempo médio de execução de cada etapa do algoritmo precisa ser medida para verificar qual é o gargalo da implementação e se o tempo de resposta será satisfatório para o volume de documentos que precisa ser processado. A referência è a implementação anterior, baseada em expressões regulares, que é bastante rápida mas cheia de erros e com capacidade extremamente limitada de diferenciação de documentos.

Apesar de uma comparação exata entre algoritmos KNN e Naïve Bayes ser extremamente trabalhosa neste caso, pode-

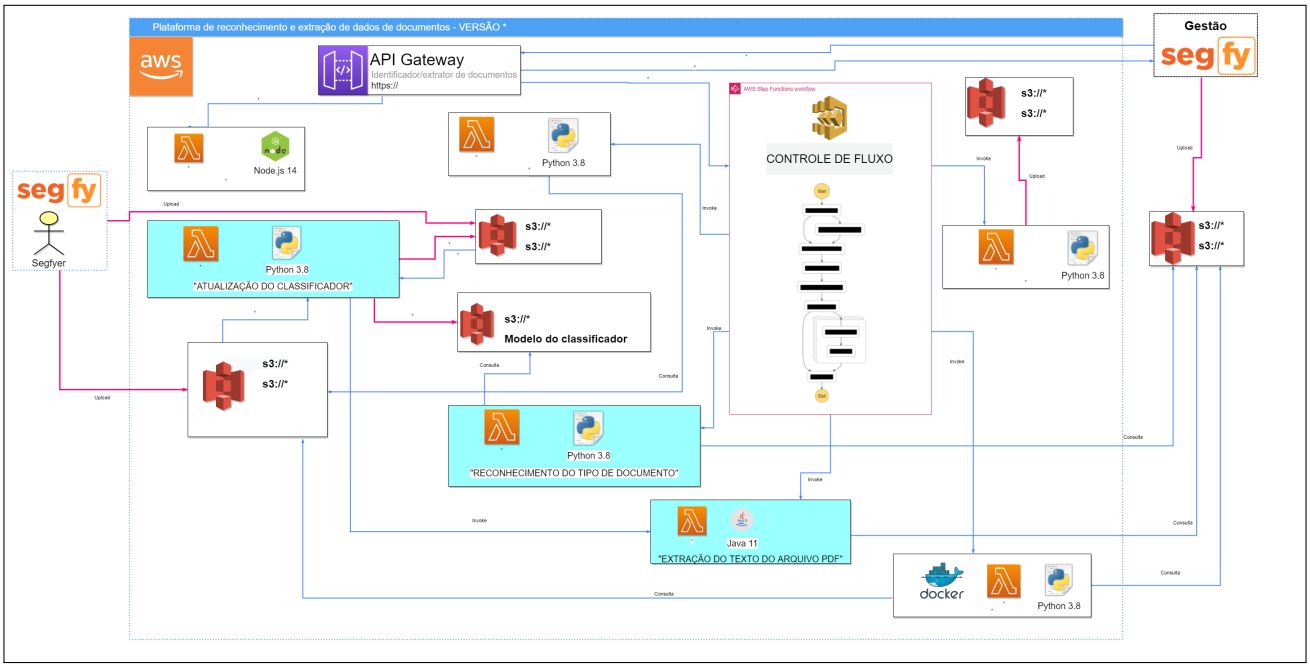


Figura 6. Arquitetura simplificada da solução de extração de dados de documentos. As partes em azul se referem à implementação do classificador, enquanto as outras estão ilustradas para compreender o contexto na solução. Nomes de objetos e estruturas de dados usadas foram omitidos por questão de segurança e sigilo. Fonte: o autor

se chegar numa análise precisa o suficiente para responder à pergunta:

> O algoritmo KNN pode superar ou igualar a acurácia do algoritmo Naïve Bayes para este tipo de problema?

É possível testar esta hipótese aplicando-se o algoritmo Naïve Bayes com as seguintes condições:

- usar o exato conteúdo que o algoritmo KNN usa para diferenciar um documento do outro. Isto oferece uma base mínima de comparação entre as abordagens, indicando o que cada algoritmo é capaz de fazer a partir dos mesmos dados;
- separar o texto em palavras e usar o TF/IDF [45] para dar peso a cada palavra e sua frequência em cada classe. Apesar desta abordagem ser ineficiente a longo prazo pelo custo computacional, é possível também estabelecer uma base de comparação razoável;
- usar exatamente a mesma base para comparar os dois algoritmos. No caso a base escolhida foi a mesma usada para verificar o valor adequado para a quantidade de *hashes* por documento, cujos resultados de comparação podem ser observados nas Figuras 10 e 11.

Há ainda mais um critério a analisar no algoritmo: seu desempenho em comparação com a solução anterior, baseada em expressoes regulares personalizadas aplicadas sobre o texto do documento inteiro. Esta abordagem tem um desempenho bastante bom (identifica o modelo em cerda de um a dois segundos), porém é bastante limitada em sua precisão:

- não identifica bem variantes dos documentos. Apenas alguns casos são corretamente identificados e a maioria dos documentos só consegue ser classificada em categorias gerais;
- exige um enorme trabalho de manutenção por parte de um especialista: é necessário comparar manualmente todos os textos de documentos similares até achar conjuntos únicos de palavras que diferenciem um tipo de documento do outro;
- exige a recompilação de uma parte expressiva da aplicação a cada inclusão de novo documento.

O classificador deve ter seu desempenho avaliado para saber se não ficará visivelmente mais lento do que a abordagem atual. Devido a isso o desempenho de todas as etapas precisa ser avaliado para confirmar se o tempo médio total não excede 5 segundos por documento, limite considerado razoável para que o usuário não perceba diferença significativa.

IV. Experimentos

Alguns experimentos foram realizados para descrever como a solução foi construída e para evidenciar a efetividade de algumas decisões tomadas na construção do classificador. Estes experimentos foram divididos da seguinte forma:

- Detalhes da implementação;
- Dados efetivamente usados;
- Modelo inicial;
- Seletividade dos *hashes*;
- Escolha da quantidade de *hashes* por documento;
- Comparação com outro algoritmo (Naïve Bayes);
- Medição do desempenho.

Cada um destes itens é detalhado a seguir.

```
{'Author': 'Registered to: * ', 'Content-Type':
"['application/pdf', 'application/xml']", 'Creation-Date': '____-
__-__T__:__:__Z', 'Last-Modified': '____-__-__T__:__:__Z', 'Last-
Save-Date': '____-__-__T__:__:__Z',
'access_permission:assemble_document': 'true',
'access_permission:can_modify': 'true',
'access_permission:can_print': 'true',
'access_permission:can_print_degraded': 'true',
'access_permission:extract_content': 'true',
'access_permission:extract_for_accessibility': 'true',
'access_permission:fill_in_form': 'true',
'access_permission:modify_annotations': 'true', 'created': '____-
__-__T__:__:__Z', 'creator': 'Registered to: * ', 'date': '____-
__-__T__:__:__Z', 'dc:creator': 'Registered to: * ', 'dc:format':
'application/pdf;', 'version': '1.3', 'dc:title': '*',
'dcterms:created': '____-__-__T__:__:__Z', 'dcterms:modified':
'____-__-__T__:__:__Z', 'meta:author': 'Registered to: * ',
'meta:creation-date': '____-__-__T__:__:__Z', 'meta:save-date':
'____-__-__T__:__:__Z', 'modified': '____-__-__T__:__:__Z',
'pdf:PDFVersion': '1.3', 'pdf:docinfo:created': '____-__-
__T__:__:__Z', 'pdf:docinfo:creator': 'Registered to: * ',
'pdf:docinfo:creator_tool': 'OpenText Exstream Version 16.6.0 64-
bit', 'pdf:docinfo:modified': '____-__-__T__:__:__Z',
'pdf:docinfo:title': '*', 'pdf:encrypted': 'false',
'pdf:hasMarkedContent': 'false', 'pdf:hasXFA': 'false',
'pdf:hasXMP': 'false', 'title': '*', 'xmp:CreatorTool': 'OpenText
Exstream Version 16.6.0 64-bit', 'xmpTPg:NPages': '5', 'Content-
Length': '15304', 'embeddedResourceType': 'ATTACHMENT'}
```


Figura 7. Exemplo de metadados usados para entrada para o classificador. Conforme citado anteriormente apenas os dados relevantes para classificação foram incluídos no texto extraído. Fonte: o autor


```
["{'/ColorSpace': '/DeviceRGB', '/Decode': [0, 1, 0, 1, 0, 1],
'/Subtype': '/Image', '/Height': 59, '/Filter': ['/FlateDecode',
'/DCTDecode'], '/Type': '/XObject', '/Width': 59,
'/BitsPerComponent': 8}", "{'/ColorSpace': '/DeviceRGB',
'/Decode': [0, 1, 0, 1, 0, 1], '/Subtype': '/Image', '/Height':
66, '/Filter': ['/FlateDecode', '/DCTDecode'], '/Type':
'/XObject', '/Width': 65, '/BitsPerComponent': 8}",
"{'/ColorSpace': '/DeviceRGB', '/Decode': [0, 1, 0, 1, 0, 1],
'/Subtype': '/Image', '/Height': 142, '/Filter': ['/FlateDecode',
'/DCTDecode'], '/Type': '/XObject', '/Width': 473,
'/BitsPerComponent': 8}", "{'/ColorSpace': '/DeviceRGB',
'/Decode': [0, 1, 0, 1, 0, 1], '/Subtype': '/Image', '/Height':
59, '/Filter': ['/FlateDecode', '/DCTDecode'], '/Type':
'/XObject', '/Width': 59, '/BitsPerComponent': 8}",
"{'/ColorSpace': '/DeviceRGB', '/Decode': [0, 1, 0, 1, 0, 1],
'/Subtype': '/Image', '/Height': 59, '/Filter': ['/FlateDecode',
'/DCTDecode'], '/Type': '/XObject', '/Width': 59,
'/BitsPerComponent': 8}", "{'/ColorSpace': '/DeviceRGB',
'/Decode': [0, 1, 0, 1, 0, 1], '/Subtype': '/Image', '/Height':
130, '/Filter': ['/FlateDecode', '/DCTDecode'], '/Type':
'/XObject', '/Width': 106, '/BitsPerComponent': 8}",
"{'/ColorSpace': '/DeviceRGB', '/Decode': [0, 1, 0, 1, 0, 1],
'/Subtype': '/Image', '/Height': 59, '/Filter': ['/FlateDecode',
'/DCTDecode'], '/Type': '/XObject', '/Width': 59,
'/BitsPerComponent': 8}", "{'/ColorSpace': '/DeviceRGB',
'/Decode': [0, 1, 0, 1, 0, 1], '/Subtype': '/Image', '/Height':
59, '/Filter': ['/FlateDecode', '/DCTDecode'], '/Type':
'/XObject', '/Width': 59, '/BitsPerComponent': 8}",
"{'/ColorSpace': '/DeviceRGB', '/Decode': [0, 1, 0, 1, 0, 1],
'/Subtype': '/Image', '/Height': 59, '/Filter': ['/FlateDecode',
'/DCTDecode'], '/Type': '/XObject', '/Width': 59,
'/BitsPerComponent': 8}", "{'/ColorSpace': '/DeviceRGB',
'/Decode': [0, 1, 0, 1, 0, 1], '/Subtype': '/Image', '/Height':
59, '/Filter': ['/FlateDecode', '/DCTDecode'], '/Type':
'/XObject', '/Width': 59, '/BitsPerComponent': 8}"]
```


Figura 8. Exemplo de metadados usados para entrada para o classificador. Nomes comerciais existentes no documento foram substituídos por (“*”) apenas na figura para não expor dados da companhia. Conforme citado anteriormente dados sabidamente irrelevantes para classificação foram automaticamente substituídos por caracteres neutros (“_”). Fonte: o autor

A. *Detalhes da implementação*

A implementação foi integrada com os demais processos de tratamento de documentos da empresa conforme a Figura 6. A parte do classificador foi dividida em 3 funções:

- ”EXTRAÇÃO DO TEXTO DO ARQUIVO PDF”: analisa o arquivo PDF, coletando os metadados sobre as propriedades do documento, localização das imagens e um trecho do texto do documento para a geração dos *hashes*;
- ”RECONHECIMENTO DO TIPO DE DOCUMENTO”: usando o texto extraído do documento, calcula os *minhashes* e junta-os para gerar o resumo do documento, que é então comparado com os resumos dos documentos do modelo em busca daqueles que tiverem a maior quantidade de *minhashes* em comum. Usando se necessário os critérios de desempate descritos anteriormente, define a classe do documento e devolve para o controle de fluxo a fim de que o tratamento adequado seja realizado;
- ”ATUALIZAÇÃO DO CLASSIFICADOR”: esta função é responsável por atualizar o modelo do classificador, acrescentando classes correspondentes aos novos modelos de documentos suportados ou complementado as classes existentes com mais exemplos de documentos. Este processo sempre é disparado internamente e de maneira assíncrona por um colaborador da empresa. Durante este processo os dados pessoais do documento são removidos para não poluir o modelo com informações identificáveis de segurados.

B. *Dados efetivamente usados*

A Figura 7 ilustra um trecho de texto processado dos metadados de um arquivo PDF. Informações de data/hora foram convertidas para um formato genérico e a ordem dos metadados foi mantida. Desta forma, os textos presentes, valores e sua ordem contribuem para identificar unicamente o modelo de documento, especialmente pelos nomes das ferramentas e números de versões.

A Figura 8 mostra outro trecho de metadados correspondente às imagens do documento. As imagens propriamente ditas são desnecessárias para a classificação, pois as características, tamanhos e ordem dos elementos foram considerados suficientes para diferenciar um modelo do outro.

Estas informações, unidas a um trecho de texto do documento, foram suficientes para diferenciar um modelo de documento de todos os outros nos experimentos realizados.

C. *Modelo inicial*

A versão preliminar do modelo, capaz apenas de identificar a companhia, foi treinada com 70% dos dados disponíveis, reservando o restante para medir a acurácia. A avaliação do modelo gerado sobre a base de testes (que não foram apresentadas ao classificador durante a fase de treinamento) obteve 98% de acurácia, indicando que o modelo seria satisfatório para a aplicação proposta uma vez que todas as classificações incorretas foram tratadas como tendo o mesmo peso.

Classe	Documentos	Incluídos no modelo
digitalizado	1	1
seg1	3	3
seg2	1	1
seg3	56	14
seg5	256	102
seg6	11	6
seg7	13	4
seg8	30	10
seg9	398	132
seg10	10	6
seg11	18	6
seg12	224	74
seg13	4	2
seg14	31	7
seg15	66	21
seg16	4	2
seg17	28	10
seg18	95	42
seg19	11	4
seg20	302	127
seg21	31	9
seg21a	-	2
seg21b	-	2
seg21c	-	1
seg21d	-	2
seg21e	-	1
seg21f	-	1
seg21g	-	2
seg21h	-	1
seg21i	-	2
seg21j	-	2
seg21k	-	1
seg21l	-	4
seg22	1	1
seg23	36	10
seg24	26	9
seg25	478	179
seg26	461	141
seg28	39	14
seg29	510	204
seg30	4	4
seg31	8	5
seg32	81	19
seg33	3	3
seg34	221	85
seg35	55	20
seg36	245	86
seg37	16	4
seg38	276	92
seg39	29	9
seg40	28	10
seg42	116	29

Tabela II
QUANTIDADE DE DOCUMENTOS USADOS PARA TREINAMENTO/TESTE E INSTÂNCIAS INCORPORADAS AO MODELO APÓS A SUBDIVISÃO DE UMA DAS COMPANHIAS. FONTE: O AUTOR

Após validar a viabilidade do modelo e sua provável acurácia para dados novos, decidiu-se manter a acurácia em 100% (dentro dos documentos conhecidos) incorporando ao classificador todos os documentos da base de testes que foram lidos incorretamente. A partir deste momento ficou definido que a acurácia efetiva do modelo seria medida sempre sobre os dados reais coletados diretamente do ambiente de produção.

Depois disso, como aplicação do classificador, escolheu-se arbitrariamente uma companhia para subdividir em categorias específicas de documentos. No caso, os documentos de uma das seguradoras (*seg21*) foram divididos em 12 tipos específicos (*seg21a* até *seg21l*).

As quantidades de documentos que foram incorporados ao modelo do classificador podem ser vistas na Tabela II. Estas quantidades foram suficientes para iniciar o uso do classificador. A política de atualização do modelo é sempre incluir qualquer documento classificado incorretamente de forma que a acurácia sobre os documentos conhecidos (ou seja, aqueles já apresentados ao classificador) mantenha-se sempre em 100%.

D. Seletividade dos hashes

A Figura 9 mostra as métricas analisadas para avaliar a eficácia dos *hashes* na classificação. As métricas analisadas foram:

- *Hashes* comuns na mesma base (A): indica a quantidade de mínima de *hashes* que é comum a todos os documentos da mesma classe. Valores altos indicam que todos os documentos da mesma classe possuem muitos trechos de texto em comum, reduzindo bastante a probabilidade de um documento ser confundido com outra classe. O valor zero acabou se tornando frequente porque a base de treinamento possui diversos tipos de documentos diferentes da mesma companhia, que a princípio não possuem trechos de texto em comum e portanto não possuem *hashes* em comum;
- Máximo de *hashes* comuns em outras classes (B): indica a maior quantidade de *hashes* em comum com um documento de classe diferente. O desejável é que a quantidade de *hashes* em comum com documentos de outras classes seja menor do que a quantidade de *hashes* comuns entre documentos da mesma classe. Valores menores indicam a situação ideal em que o *hash* é suficiente para classificar corretamente um novo documento;
- *Hashes* distintos dentro da classe (C): quantidade de *hashes* diferentes encontrados nos documentos pertencentes à mesma classe de documentos. O desejável é que estes *hashes* sejam encontrados apenas dentro de documentos da mesma classe;
- *Hashes* comuns com outras classes (D): quantidade de *hashes* que estão presentes tanto em documentos de mais de uma classe. Os *hashes* em comum não são suficientes para individualmente identificar a classe à qual um documento pertence. Quanto menor este número, menor é a quantidade de texto em comum com outras classes de documentos e consequentemente maior a probabilidade do *hash* ser suficiente para identificar a classe do documento;
- % de *hashes* comuns (E): indica qual o percentual de *hashes* dentro de uma classe que é comum a mais de uma classe de documentos. É a razão entre as colunas (D) e (C);
- Probabilidade de empate (F): indica a probabilidade de um documento de outra classe ter um número maior ou igual de *hashes* em comum do que um documento da classe correta. A princípio só pode ocorrer nos casos em que um documento de outra classe tiver mais *hashes* em

Classe	Documentos	Hashes comuns na mesma classe (A)	Máximo de hashes comuns em outras classes (B)	Hashes distintos dentro da classe (C)	Hashes comuns com outras classes (D)	% de hashes comuns (E)	Probabilidade de empate (máxima) (F)	Probabilidade de uso do desempate (G)
digitalizado	1	20	2	20	5	25,00%	0,00%	0,00%
seg1	3	8	3	41	10	24,39%	0,00%	0,00%
seg2	1	20	1	20	4	20,00%	0,00%	0,00%
seg3	14	0	4	205	18	8,78%	1,95%	0,17%
seg5	102	0	4	597	50	8,38%	0,67%	0,06%
seg6	6	1	4	76	8	10,53%	3,95%	0,42%
seg7	4	7	1	36	3	8,33%	0,00%	0,00%
seg8	10	0	4	137	19	13,87%	2,92%	0,40%
seg9	132	0	11	691	63	9,12%	1,59%	0,15%
seg10	6	9	2	49	5	10,20%	0,00%	0,00%
seg11	6	0	2	82	12	14,63%	2,44%	0,36%
seg12	74	0	4	734	55	7,49%	0,54%	0,04%
seg13	2	10	2	30	5	16,67%	0,00%	0,00%
seg14	7	0	2	74	11	14,86%	2,70%	0,40%
seg15	21	0	3	149	17	11,41%	2,01%	0,23%
seg16	2	3	3	37	5	13,51%	0,00%	0,00%
seg17	10	0	2	98	10	10,20%	2,04%	0,21%
seg18	42	0	5	305	32	10,49%	1,64%	0,17%
seg19	4	0	2	44	10	22,73%	4,55%	1,03%
seg20	127	0	6	691	51	7,38%	0,87%	0,06%
seg21	9	0	16	58	26	44,83%	27,59%	12,37%
seg21a	2	6	2	34	8	23,53%	0,00%	0,00%
seg21b	2	10	5	30	12	40,00%	0,00%	0,00%
seg21c	1	20	1	20	3	15,00%	0,00%	0,00%
seg21d	2	15	1	25	2	8,00%	0,00%	0,00%
seg21e	1	20	3	20	8	40,00%	0,00%	0,00%
seg21f	1	20	2	20	4	20,00%	0,00%	0,00%
seg21g	2	15	2	25	6	24,00%	0,00%	0,00%
seg21h	1	20	1	20	3	15,00%	0,00%	0,00%
seg21i	2	15	1	25	2	8,00%	0,00%	0,00%
seg21j	2	15	16	25	18	72,00%	4,00%	2,88%
seg21k	1	20	7	20	9	45,00%	0,00%	0,00%
seg21l	4	4	5	43	16	37,21%	2,33%	0,87%
seg22	1	20	4	20	5	25,00%	0,00%	0,00%
seg23	10	0	10	142	45	31,69%	7,04%	2,23%
seg24	9	0	6	58	14	24,14%	10,34%	2,50%
seg25	179	0	6	771	41	5,32%	0,78%	0,04%
seg26	141	0	5	1231	86	6,99%	0,41%	0,03%
seg28	14	0	3	154	27	17,53%	1,95%	0,34%
seg29	204	0	11	1544	123	7,97%	0,71%	0,06%
seg30	4	9	2	43	5	11,63%	0,00%	0,00%
seg31	5	3	1	50	3	6,00%	0,00%	0,00%
seg32	19	0	4	183	16	8,74%	2,19%	0,19%
seg33	3	11	1	34	2	5,88%	0,00%	0,00%
seg34	85	0	4	631	38	6,02%	0,63%	0,04%
seg35	20	0	4	186	27	14,52%	2,15%	0,31%
seg36	86	0	4	490	31	6,33%	0,82%	0,05%
seg37	4	6	2	43	5	11,63%	0,00%	0,00%
seg38	92	0	5	695	70	10,07%	0,72%	0,07%
seg39	9	0	6	59	15	25,42%	10,17%	2,59%
seg40	10	0	2	77	10	12,99%	2,60%	0,34%
seg42	29	0	6	346	40	11,56%	1,73%	0,20%

Figura 9. Análise da eficácia dos *hashes* para identificar a classe do documento. Fonte: o autor

comum com um documento do que os documentos da classe correta. É dado pela fórmula:

$$\begin{cases} \frac{B-A}{C} & \text{quando for positivo} \\ 0, & \text{caso contrário} \end{cases}$$

- Probabilidade de necessidade de desempate (G): estimativa (pessimista) de quando ocorrerá a identificação de *hashes* comuns tanto em documento dentro da classe quanto fora e a identificação de um documento de outra classe com o mesmo número de *hashes* em comum do que com um documento da classe correta (exigindo a comparação integral do texto). É o produto das colunas (E) e (F).

Como se pode observar nos valores, na maior parte dos casos a comparação de *hashes* é suficiente para identificar corretamente a classe de um documento, sem a necessidade de comparação textual (desempate).

E. Escolha da quantidade de hashes por documento

Apenas para ilustrar a eficácia do valor escolhido ilustram-se alguns casos usando uma amostra de 1475 documentos. A Figura 10 mostra o melhor caso encontrado, com 5 erros, enquanto a Figura 11 mostra o efeito de dobrar a quantidade de *hashes*, que não mostrou benefício na acurácia, ficando com 14 erros. Os valores testados e acurácias respectivas estão na tabela III. Como se pode verificar, a acurácia em todos os casos ficou abaixo de 1%, valor aceitável para a aplicação no momento.

F. Comparação com outro algoritmo (Naïve Bayes)

O resultado da execução do algoritmo Naïve Bayes nas condições propostas anteriormente é apresentado na Figura 12. A quantidade de erros foi bem maior (404), levando a uma acurácia de apenas 72%. Conforme ilustrado anteriormente, muitos documentos possuem praticamente as mesmas

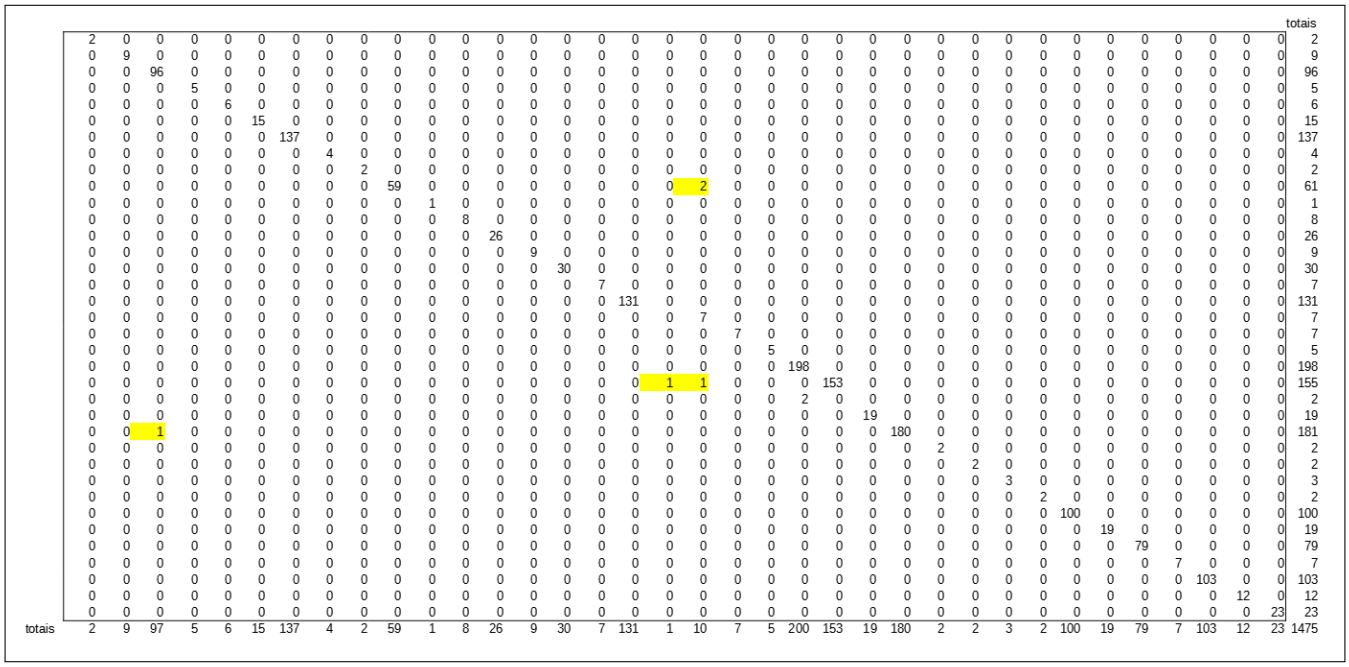


Figura 10. Matriz de confusão usando 20 *hashes* por documento. Os erros estão destacados para facilitar a visualização. As linhas contendo apenas zeros foram eliminadas. Fonte: o autor

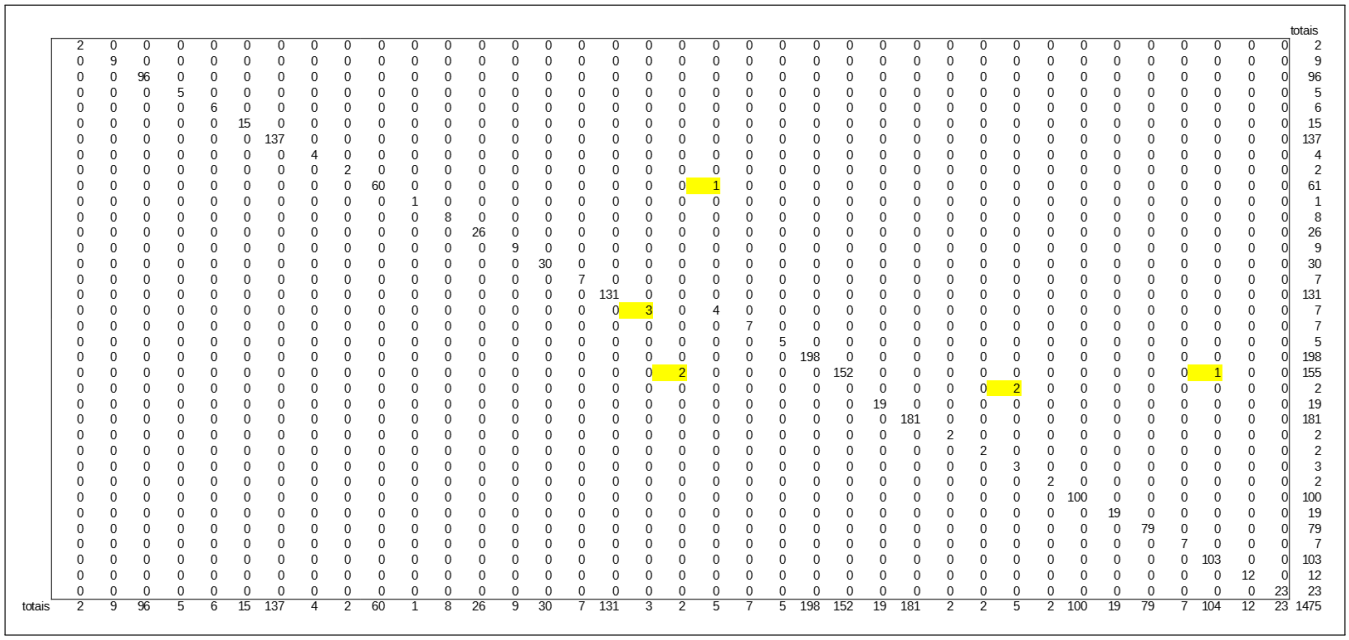
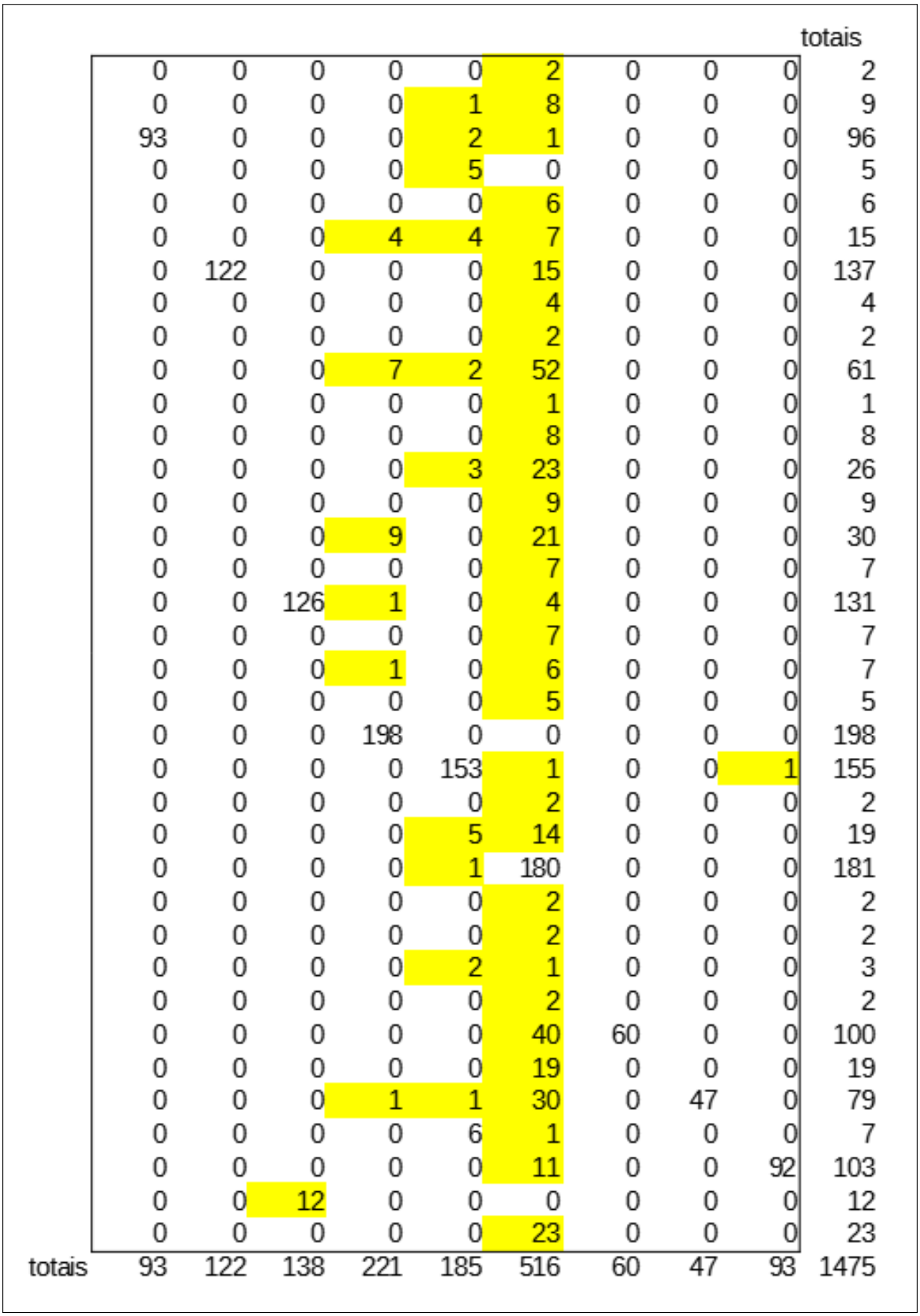


Figura 11. Matriz de confusão usando 40 *hashes* por documento. Os erros estão destacados para facilitar a visualização. As linhas contendo apenas zeros foram eliminadas. Fonte: o autor



										totais
	0	0	0	0	0	2	0	0	0	2
	0	0	0	0	1	8	0	0	0	9
	93	0	0	0	2	1	0	0	0	96
	0	0	0	0	5	0	0	0	0	5
	0	0	0	0	0	6	0	0	0	6
	0	0	0	4	4	7	0	0	0	15
	0	122	0	0	0	15	0	0	0	137
	0	0	0	0	0	4	0	0	0	4
	0	0	0	0	0	2	0	0	0	2
	0	0	0	7	2	52	0	0	0	61
	0	0	0	0	0	1	0	0	0	1
	0	0	0	0	0	8	0	0	0	8
	0	0	0	0	3	23	0	0	0	26
	0	0	0	0	0	9	0	0	0	9
	0	0	0	9	0	21	0	0	0	30
	0	0	0	0	0	7	0	0	0	7
	0	0	126	1	0	4	0	0	0	131
	0	0	0	0	0	7	0	0	0	7
	0	0	0	1	0	6	0	0	0	7
	0	0	0	0	0	5	0	0	0	5
	0	0	0	198	0	0	0	0	0	198
	0	0	0	0	153	1	0	0	1	155
	0	0	0	0	0	2	0	0	0	2
	0	0	0	0	5	14	0	0	0	19
	0	0	0	0	1	180	0	0	0	181
	0	0	0	0	0	2	0	0	0	2
	0	0	0	0	0	2	0	0	0	2
	0	0	0	0	2	1	0	0	0	3
	0	0	0	0	0	2	0	0	0	2
	0	0	0	0	0	40	60	0	0	100
	0	0	0	0	0	19	0	0	0	19
	0	0	0	1	1	30	0	47	0	79
	0	0	0	0	6	1	0	0	0	7
	0	0	0	0	0	11	0	0	92	103
	0	0	12	0	0	0	0	0	0	12
	0	0	0	0	0	23	0	0	0	23
totais	93	122	138	221	185	516	60	47	93	1475

Figura 12. Matriz de confusão do algoritmo Naïve Bayes sobre a mesma base de testes das Figuras 10 e 11. Os 404 erros estão destacados para facilitar a visualização. As linhas contendo apenas zeros foram eliminadas. Fonte: o autor

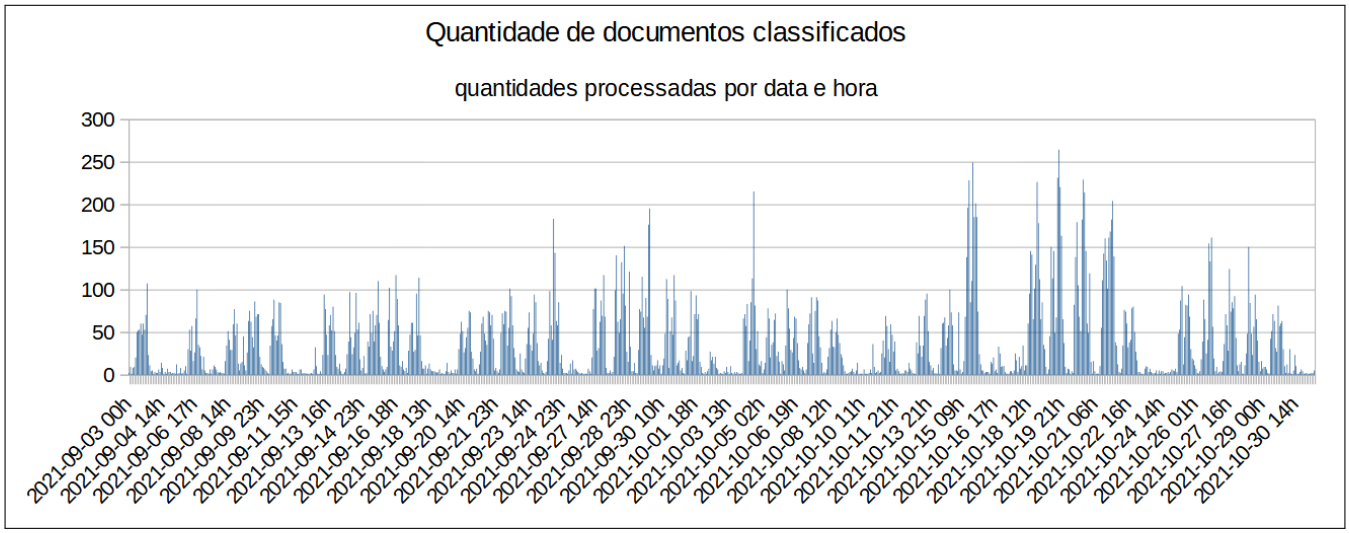


Figura 13. Volume de documentos classificados por hora em um período de 60 dias. Quantidade representam apenas uma fração dos documentos processados de 3 companhias. Fonte: o autor

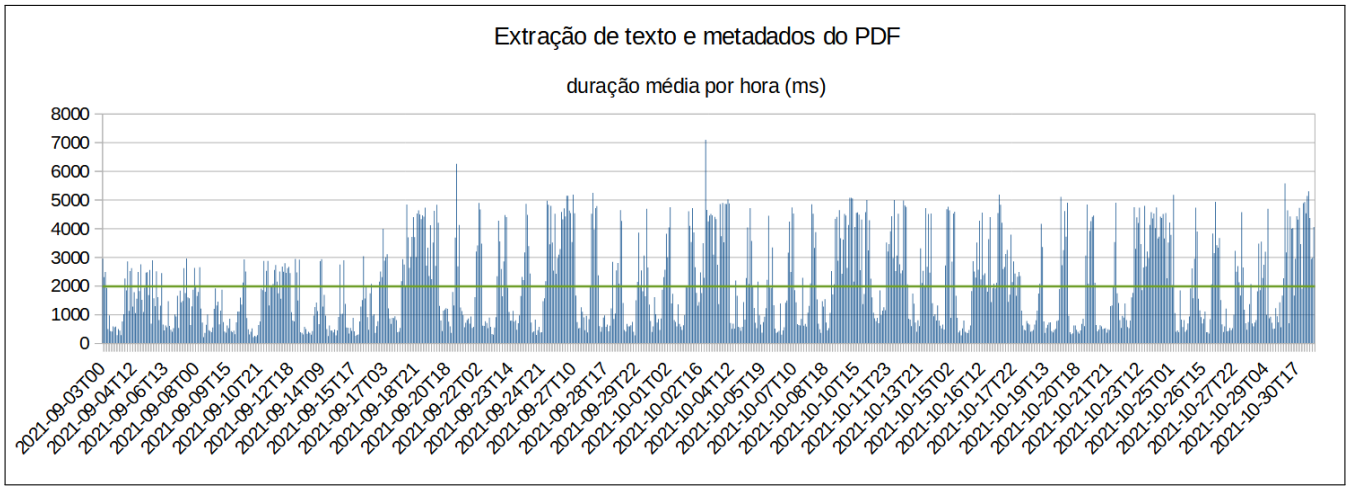


Figura 14. Tempo médio de extração de metadados e texto dos documentos classificados, analisados de hora em hora durante um período de 60 dias. A linha verde representa a média de todos os valores apresentados, que é de 1991 milissegundos. Fonte: o autor

Número de *hashes*	Erros	Acurácia
20	5	0,33%
25	14	0,95%
30	12	0,81%
40	14	0,95%

Tabela III
ACURÁCIA DE ALGUNS VALORES TESTADOS PARA A QUANTIDADE DE *hashes* POR DOCUMENTO. FONTE: O AUTOR

palavras, apenas em outra ordem. Mesmo incluindo o texto dos metadados, apenas a lista de palavras não se mostra suficiente para chegar a um bom resultado. Outro efeito evidente é a tendência de classificar os documentos de forma proporcional à ocorrência por classe. Como há poucas classes com grande quantidade de documentos, a tendência do algoritmo é não classificar documento algum nas classes menos frequentes, fazendo com que apenas as classes com mais exemplos de teste sejam usadas. Este efeito precisaria de um tratamento bastante extenso para deixar todas as classes com aproximadamente a mesma quantidade de dados de treinamento e isso simplesmente não é viável para o conjunto de documentos em questão. Outros critérios de classificação ou agrupamento também não foram considerados porque o objetivo da separação é escolher o extrator certo para cada leiaute de documento.

Voltando-se à pergunta sobre a comparação entre os algoritmos, observa-se que a acurácia do algoritmo KNN (acima de 99%) foi bem superior à do algoritmo Naïve Bayes no contexto indicado, e portanto mesmo com as otimizações que se poderia fazer no Naïve Bayes para aprimorá-lo a diferença demonstra que a implementação via KNN é bastante viável e seria capaz de igualar ou superar o Naïve Bayes no contexto do problema proposto.

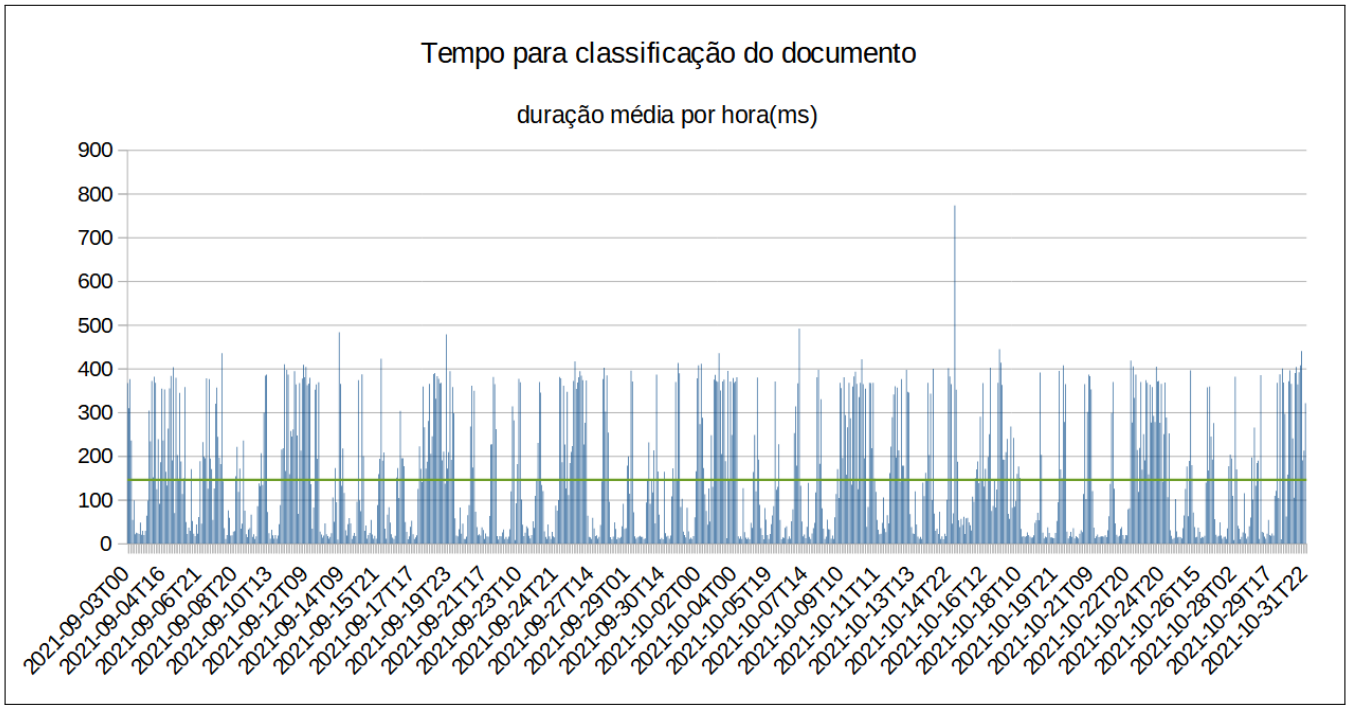


Figura 15. Tempo médio de classificação dos documentos, analisados de hora em hora durante um período de 60 dias. A linha verde representa a média de todos os valores apresentados, que é de 146 milissegundos. Fonte: o autor

G. Medição do desempenho

A Figura 13 apresenta o volume de documentos processados pelo classificador em um período de 60 dias, agrupados por hora. A solução foi implementada em várias etapas, conforme a demanda e o surgimento de novas versões de documentos para as companhias escolhidas para adotar a solução. O fluxo varia bastante conforme o horário e dia da semana, pois muitas corretoras organizam as atividades por dia da semana.

O volume processado no período chega a 264 documentos por hora, o que representa apenas uma pequena fração do volume de cada uma das companhias escolhidas. O critério de adoção é lento e incremental porque já existe um outro método menos eficiente de extração de dados para a maioria dos documentos e o desempenho de todos os componentes da solução possa ser medido em custo, desempenho e relevância para os clientes.

Sobre o volume indicado na Figura 13, foram feitas medições do tempo gasto no processamento das duas etapas da classificação:

- (A) Extração de texto e metadados do documento PDF, incluindo a eventual descriptografia dos dados (no caso da versão 6.0 do formato de arquivo PDF), remoção de dados irrelevantes para classificação e padronização dos metadados;
- (B) Uso do texto extraído para criação dos *hashes* e comparação com todos os outros documentos que fazem parte da base de classificação, com a aplicação dos eventuais critérios de desempate.

O resultado de desempenho para a função em nuvem que executa a parte (A) de extração de texto pode ser observada na Figura 14. A variação é bastante grande no tempo porque, pela natureza da execução em nuvem, a função só é mantida instanciada em memória enquanto há um fluxo constante de requisições, o que não acontece durante os períodos de baixa demanda e porque a quantidade de documentos enviados para classificação ainda é muito pequena. O maior tempo (7 segundos) ocorreu durante as 21h de um sábado, que foi a única solicitação durante a hora. O tempo de partida a frio ainda é relativamente grande mas não chega a ser um problema porque só ocorre em períodos de baixa utilização.

A variação no volume de processamento também é bastante grande porque os documentos possuem tamanhos extramemente variados, indo de pouco mais de 32 kilobytes até 13 megabytes, dependendo do tipo de documento e da companhia. Durante os períodos de alta demanda, com a função já instanciada em memória, o tempo de resposta para a operação de extração de metadados ficou frequentemente abaixo de um segundo, com mínimo de 205 milissegundos. Este tempo foi considerado perfeitamente aceitável para a extração de metadados e não mudou significativamente em relação à solução anterior.

Quantos aos tempos de execução da função de classificação do documento propriamente dita, que usa apenas os metadados e textos da etapa anterior, os resultados estão na Figura 15. Novamente observa-se grande variação nas médias dependendo do horário de uso da função, porém com um tempo de execução muito menor. Calcular todos os *hashes* e compará-los com todos os outros de todas as classes (incluindo eventuais comparações textuais de desempate) consumiu em

média apenas 146 milissegundos, demonstrando que a solução adotada é equivalente à solução anterior (baseada em expressões regulares e texto) quanto a desempenho e muito mais precisa quanto à identificação do documento, o que permite o uso de extratores muito mais detalhados e confiáveis, capazes de oferecer muito mais segurança e qualidade no tratamento dos dados.

V. Conclusões e trabalhos futuros

A solução atendeu a todos os requisitos de acurácia e desempenho propostos durante o período de 60 dias usado como referência. Além disso, o nível de diferenciação entre documentos superou as abordagens anteriores, permitindo diferenciação de tipos muito mais detalhados e que permitem extração de dados muito mais completa. Apesar disso, ainda há muitas coisas que podem ser melhoradas na implementação:

- A separação da base em treinamento e testes pode ser feita por um processo aleatório que garanta a proporcionalidade de classes de cada categoria, facilitando assim a comparação com outros algoritmos;
- A acurácia é uma métrica muito simples. Pode-se analisar o algoritmo sobre outras métricas mais elaboradas, como as medidas F ou a mais popular delas, a medida F1 [43];
- Usar outros algoritmos e abordagens para comparação, como por exemplo redes neurais. Mesmo tendo o problema do tempo de treinamento, a rede neural treinada teoricamente teria um desempenho muito melhor na classificação do que o algoritmo KNN;
- O KNN se baseia na comparação do documento a classificar com todos os documentos do modelo. Apesar da comparação envolver apenas hashes, ainda assim a quantidade de documentos tende a crescer e o consumo de memória e processador também. Pode-se buscar alguma forma de manter o bom desempenho considerando o crescimento da base.

Referências

[1] R. Sharda, D. Delen, and E. Turban, *Business Intelligence e Análise de Dados para Gestão do Negócio - 4.ed.* Bookman Editora, 2019. [Online]. Available: https://books.google.com.br/books?id=Qr6xDwAAQBAJ

[2] R. Vasilescu, "Pdf/a standard for long term archiving," 2009.

[3] W. McKnight, "Text data mining in business intelligence," *Information Management Magazine*, vol. 15, no. 1, 2005.

[4] J. Tiedemann, "Improved text extraction from pdf documents for large-scale natural language processing," in *Computational Linguistics and Intelligent Text Processing*, A. Gelbukh, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 102–112.

[5] F. M. Matsubara, T. Hanada, S. Imai, S. Miura, and S. Akatsu, "Managing a media server content directory in absence of reliable metadata," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 873–877, 2009.

[6] J. R. Hobbs and E. Riloff, "Information extraction." *Handbook of natural language processing*, vol. 15, p. 16, 2010.

[7] D. Corrigan, D. deRoos, T. Deutsch, K. Parasuraman, P. Zikopoulos, and J. Giles, *Harness the Power of Big Data The IBM Big Data Platform*. McGraw-Hill Osborne Media, 2012. [Online]. Available: https://mhebooklibrary.com/doi/book/10.1036/9780071808187

[8] M. Ling, J. Chen, T. Möller, P. Isenberg, T. Isenberg, M. Sedlmair, R. S. Laramee, H. Shen, J. Wu, and C. L. Giles, "Document domain randomization for deep learning document layout extraction," *CoRR*, vol. abs/2105.14931, 2021. [Online]. Available: https://arxiv.org/abs/2105.14931

[9] Brasil, "Lei geral de proteção de dados pessoais (lgpd)," *Diário Oficial da República Federativa do Brasil*, 2018. [Online]. Available: http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm

[10] J. Voerman, A. Joseph, M. Coustaty, V. P. D'Andecy, and J. Ogier, "Evaluation of neural network classification systems on document stream," *CoRR*, vol. abs/2007.07547, 2020. [Online]. Available: https://arxiv.org/abs/2007.07547

[11] P. C. Sen, M. Hajra, and M. Ghosh, "Supervised classification algorithms in machine learning: A survey and review," in *Emerging Technology in Modelling and Graphics*, J. K. Mandal and D. Bhattacharya, Eds. Singapore: Springer Singapore, 2020, pp. 99–111.

[12] B. Maake and T. Zuva, "A comparative analysis of text similarity measures and algorithms in research paper recommender systems," in *ICTAS*, 03 2018.

[13] A. Dhar, H. Mukherjee, N. S. Dash, and K. Roy, "Text categorization: past and present," *Artificial Intelligence Review*, vol. 54, no. 4, pp. 3007–3054, Apr 2021. [Online]. Available: https://doi.org/10.1007/s10462-020-09919-1

[14] "Segfy tecnologia," https://www.segfy.com, acessado: 2021-11-02.

[15] A. Khan, B. Bahurdin, and K. Khan, "An overview of e-documents classification," *Proceedings of International Conference on Machine Learning and Computing (ICMLC 2009)*, Jan 2009.

[16] C. Clark and S. Divvala, "Pdffigures 2.0: Mining figures from research papers," in *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries*, ser. JCDL '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 143–152. [Online]. Available: https://doi.org/10.1145/2910896.2910904

[17] G. M. Binmakhashen and S. A. Mahmoud, "Document layout analysis: A comprehensive survey," *ACM Comput. Surv.*, vol. 52, no. 6, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3355610

[18] S. S. Budhiraja and V. Mago, "Extracting learning outcomes using machine learning and white space analysis," in *Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good*, ser. Goodtechs '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 7–12. [Online]. Available: https://doi.org/10.1145/3284869.3284879

[19] J. Beel, S. Langer, M. Genzmehr, and C. Müller, "Docear's pdf inspector: Title extraction from pdf files," in *Proceedings of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries*, ser. JCDL '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 443–444. [Online]. Available: https://doi.org/10.1145/2467696.2467789

[20] "Amazon textract features," https://aws.amazon.com/textract/features, acessado: 2021-10-02.

[21] "Pdf scraper for businesses," https://nanonets.com/pdf-scraper, acessado: 2021-10-02.

[22] "Document ai," https://cloud.google.com/solutions/procurement-doc-ai, acessado: 2021-10-02.

[23] "Amazon comprehend - custom classification," https://docs.aws.amazon.com/comprehend/latest/dg/how-document-classification.html, acessado: 2021-10-02.

[24] "Automl - preparing your training data," https://cloud.google.com/natural-language/automl/docs/prepare, acessado: 2021-10-02.

[25] R. Sharda, D. Delen, and E. Turban, *Business Intelligence e Análise de Dados para Gestão do Negócio - 4.ed.* Bookman Editora, 2019, ch. 5. [Online]. Available: https://books.google.com.br/books?id=Qr6xDwAAQBAJ

[26] J. Grus, *Data Science do Zero*. Editora Alta Books, 2021, ch. 21. [Online]. Available: https://integrada.minhabiblioteca.com.br/books/9788550816463

[27] V. K. Vijayan, K. R. Bindu, and L. Parameswaran, "A comprehensive study of text classification algorithms," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 1109–1113.

[28] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, July 2008, ch. Vector space classification. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0521865719

[29] ——, *Introduction to Information Retrieval*. Cambridge University Press, July 2008, ch. Text classification and Naive Bayes. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0521865719

[30] ——, *Introduction to Information Retrieval*. Cambridge University Press, July 2008, ch. Support vector machines and machine learning

on documents. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0521865719

[31] J. Zhang and Y. Yang, "Robustness of regularized linear classification methods in text categorization," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, ser. SIGIR '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 190–197. [Online]. Available: https://doi.org/10.1145/860435.860471

[32] A. Ciapetti, R. Di Florio, L. Lomasto, G. Miscione, G. Ruggiero, and D. Toti, "Nethic: A system for automatic text classification using neural networks and hierarchical taxonomies," in *Proceedings of the 23rd International Conference on Enterprise Information Systems*, 01 2019, pp. 296–306.

[33] N. Silva, F. Braz, and T. de Campos, "Document type classification for brazil's supreme court using a convolutional neural network," in *ICoFCS-2018*, 10 2018, pp. 7–11.

[34] M. Z. Afzal, S. Capobianco, M. I. Malik, S. Marinai, T. Breuel, A. Dengel, and M. Liwicki, "Deepdocclassifier: Document classification with deep convolutional neural network," *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1111–1115, 2015.

[35] D. Prasetya, A. Wibawa, and T. Hirashima, "The performance of text similarity algorithms," *International Journal of Advances in Intelligent Informatics*, vol. 4, 05 2018.

[36] Z. E. Rasjid and R. Setiawan, "Performance comparison and optimization of text document classification using k-nn and naïve bayes classification techniques," *Procedia Computer Science*, vol. 116, pp. 107–112, 2017, discovery and innovation of computer science technology in artificial intelligence era: The 2nd International Conference on Computer Science and Computational Intelligence (ICCSCI 2017). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917320549

[37] M. Slaney and M. Casey, "Locality-sensitive hashing for finding nearest neighbors [lecture notes]," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 128–131, 2008.

[38] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*, 3rd ed. USA: Cambridge University Press, 2020, ch. 3.

[39] "Create a classifier in microsoft sharepoint syntex," https://docs.microsoft.com/en-us/microsoft-365/contentunderstanding/create-a-classifier, acessado: 2021-10-02.

[40] "Apache pdfbox® - a java pdf library." [Online]. Available: https://pdfbox.apache.org/

[41] E. Giannatou, "Document-similarity-k-shingles-minhashing-lsh," https://github.com/evagian/Document-similarity-K-shingles-minhashing-LSH-python, 2017.

[42] D. V. Sarwate, "Computation of cyclic redundancy checks via table look-up," *Commun. ACM*, vol. 31, no. 8, p. 1008–1013, Aug. 1988. [Online]. Available: https://doi.org/10.1145/63030.63037

[43] K. Faceli, *Inteligência artificial: uma abordagem de aprendizado de máquina*. Grupo Gen - LTC, 2011, ch. 10. [Online]. Available: https://books.google.com.br/books?id=4DwelAEACAAJ

[44] R. Baeza-Yates and B. Ribeiro-Neto, *Recuperação de informação: conceitos e tecnologia das máquinas de busca*, 2nd ed. Bookman, 2013, ch. 5. [Online]. Available: https://integrada.minhabiblioteca.com.br/books/9788582600498

[45] ——, *Recuperação de informação: conceitos e tecnologia das máquinas de busca*, 2nd ed. Bookman, 2013, ch. 2. [Online]. Available: https://integrada.minhabiblioteca.com.br/books/9788582600498

[46] M. A. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida," *Journal of the American Statistical Association*, vol. 84, no. 406, pp. 414–420, 1989. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1989.10478785

[47] F. R. SILVA, J. A. SOARES, M. S. SERPA, and AL., *Cloud Computing*. Porto Alegre: SAGAH, 2020, ch. IaaS (infraestrutura como serviço). [Online]. Available: https://integrada.minhabiblioteca.com.br/books/9786556900193

[48] "Amazon ec2," https://aws.amazon.com/ec2/, acessado: 2021-11-02.

[49] F. R. SILVA, J. A. SOARES, M. S. SERPA, and AL., *Cloud Computing*. Porto Alegre: SAGAH, 2020, ch. Vantagens de cloud computing. [Online]. Available: https://integrada.minhabiblioteca.com.br/books/9786556900193