UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Décio Moritz Júnior

CLASSIFICAÇÃO DE EDITAIS LICITATÓRIOS EM ÁREAS DE ATUAÇÃO BASEADO EM APRENDIZADO SUPERVISIONADO

Florianópolis

2018

Décio Moritz Júnior

CLASSIFICAÇÃO DE EDITAIS LICITATÓRIOS EM ÁREAS DE ATUAÇÃO BASEADO EM APRENDIZADO SUPERVISIONADO

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciência da Computação para a obtenção do Grau de Bacharel em Ciência da Computação.
Orientador: Profa. Dra. Jerusa Marchi

Florianópolis

2018

RESUMO

Existem atualmente diversos serviços no Brasil que atuam fornecendo informativos de licitação, tal como a empresa Liciexpress. Essas empresas atuam fornecendo para seus clientes informações sobre editais de licitação publicados em meios oficiais e que são de interesse desses clientes. Dentro do fluxo de trabalho dessas empresas, cada edital publicado é avaliado individualmente por operadores humanos para decidir se tal edital é de interesse de um determinado cliente e este edital é classificado em uma dada categoria. Neste trabalho será construído um classificador automático de editais, baseado em técnicas de aprendizagem de máquina, visando automatizar o processo de classificação dos editais. O problema foi modelado como um *Document Classification Problem*, permitindo aplicação de técnicas de aprendizagem de máquina sobre dados textuais. Serão implementadas e testadas técnicas/algoritmos tais como Support Vector Machine, Random Forest e Redes Neurais Artificiais. O treino e avaliação das técnicas foi feito utilizando um conjunto de editais pré-classificados proveniente da empresa Liciexpress. A técnica que obteve melhor resultado dentre as que foram implementadas obteve um *score* de 88% nas métricas definidas.

Palavras-chave: Classificação de documentos, Machine Learning, Licitação.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O governo brasileiro, ao necessitar da contratação de um serviço ou compra de algum produto, o faz através de um procedimento administrativo denominado **Licitação**.

Segundo a lei de acesso à informação, e por motivos de transparência e justiça, para cada licitação deve ser publicado um **Edital** contendo as informações da licitação, permitindo que as empresas avaliem se aquele edital se aplica às suas respectivas áreas de atuação e se é interessante participar do processo licitatório. Esse edital deve ser publicado em um meio de informação acessível ao público geral, podendo ser um site mantido por algum órgão público, ou até jornais virtuais ou impressos. Existindo também casos de licitações que são publicadas apenas em jornais impressos de circulação local.

Toda e qualquer empresa brasileira poderia beneficiar-se ao participar de processos licitatórios, partindo do pressuposto que o governo é um cliente confiável e que ao participar se está contribuindo direta ou indiretamente para com a sociedade. Porém os editais são publicados em diversas fontes diferentes, às vezes de difícil acesso. Para que uma empresa tenha conhecimento de todos os possíveis processos licitatórios que estaria apta a participar teria que realizar um grande esforço buscando e analisando centenas de sites de órgãos públicos de todo o Brasil e outras centenas de jornais.

A fim de facilitar o acesso à essas informações, existem serviços que fornecem essas informações já filtradas, como a Liciexpress[1]. Parte do esforço empreendido por esses serviços está na classificação desses editais em segmentos de atuação, a fim de poder filtrar para cada empresa o conjunto de licitações que lhe são pertinentes. Grande parte desses serviços efetua a classificação de forma manual, acarretando alto custo e com grande possibilidade de erros.

Este trabalho busca uma abordagem de classificação automática, utilizando técnicas de aprendizagem de máquina, que poderia reduzir drasticamente o custo e os erros de classificação.

[1] http://www.liciexpress.com.br

1.1 CONTEXTUALIZAÇÃO

Existem atualmente diversos serviços brasileiros que atuam fornecendo informativos de licitação, tais como Liciexpress[2], ConLicitação[3], Licitacao.net[4], dentre muitos outros.

Essas empresas atuam fornecendo para seus clientes informações sobre editais de licitação que são de interesse desses clientes. Em geral são compilados os editais publicados recentemente e selecionados para cada cliente, de modo que cada um receba só aquilo que é interessante à ele.

Grande parte desses editais é obtida a partir da execução de agentes automatizados, conhecidos como *webcrawlers*, que são softwares que navegam por um site ou sistema de forma sistematizada buscando informações de interesse. Após as informações de cada edital serem coletadas, o edital precisa ser classificado.

O foco deste trabalho é a classificação automática dos editais de licitação em categorias pré-definidas que representam a área de atuação de empresas. Considerado um conjunto de categorias, que são constantes pré-definidas obtidas de um serviço real de informativos de licitação.

Os *crawlers* trazem as informações referentes ao edital de forma estruturada, sendo que o texto que define o objeto do edital (normalmente um parágrafo curto) é o texto que será levado em conta para classificá-lo. Logo, cada entrada do conjunto dados utilizado é composta por um texto simples que resume o edital e sua categoria.

O problema pode ser modelado como um *Document Classification Problem* (problema de Classificação de Documentos, em português), onde dado um conjunto de licitações, deseja-se atribuir a cada uma delas uma única categoria. Busca-se utilizar como critério de classificação técnicas já documentadas e utilizadas em aplicações reais a fim de verificar qual se aplica melhor à este domínio.

Para fins de teste e implementação das técnicas será utilizada uma base de dados real fornecida por um serviço de informativos de licitação.

[2] www.liciexpress.com.br
[3] www.conlicitacao.com.br
[4] www.licitacao.net

1.2 OBJETIVOS

O objetivo deste trabalho é construir um classificador para os editais de licitação a partir do conjunto de dados fornecido, de forma a automatizar o processo de classificação.

1.2.1 Objetivos Específicos

São objetivos específicos deste trabalho:

- Modelar o problema como um *Document Classification Problem*, detalhado na seção 2.5.2
- Selecionar técnicas aplicáveis ao *Document Classification Problem*, detalhado na seção 2.5.3
- Aplicar tais técnicas sobre uma base de dados real fornecida pela empresa Liciexpress[5], detalhado na seção 3.3
- Obter *precision*, *recall* e *f1-score* do classificador de pelo menos 80%, detalhado na seção 4.3

1.3 APRESENTAÇÃO DO TRABALHO

Neste trabalho serão abordados além do problema proposto alguns tópicos introdutórios necessários para compreensão da solução do problema. Inicialmente, no capítulo 2 serão dadas definições de Machine Learning, uma visão geral sobre os tipos de problemas de classificação para fins de contextualizar o leitor.

Serão dados detalhes acerca da classe do problema em que este trabalho se enquadra. A partir daí será detalhado o processo de construção da solução. Para maior detalhamento serão explicados os algoritmos utilizados na solução e serão apresentadas maneiras de avaliar a qualidade de uma dada solução.

E por fim o capítulo 3 detalha a implementação da proposta e os resultados dos experimentos são apresentados no capítulo 4.

[5] http://www.liciexpress.com.br

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Neste capítulo são apresentados os principais conceitos e técnicas aplicadas neste trabalho. Na seção 2.1 é dada a definição de Machine Learning e suas subdivisões, posteriormente na seção 2.2 é falado sobre problemas de classificação, que é a classe de problemas de interesse neste trabalho. Finalmente na seção 2.3 é dada uma definição mais formal sobre o problema de classificação de documentos, problema-alvo do trabalho.

A seção 2.4 apresenta algumas aplicações de classificação de documentos. Na seção 2.5 são apresentadas as etapas presentes na construção de um classificador de documentos, na seção 2.5.3 são apresentadas técnicas aplicáveis ao treinamento desse classificador e finalmente na seção 2.5.4 são apresentadas métricas para avaliação da qualidade do classificador.

Por fim na seção 2.6 são apresentados alguns trabalhos relacionados que utilizam as técnicas mencionadas neste capítulo em problemas similares.

2.1 MACHINE LEARNING

Subcampo da inteligência artificial, Machine Learning consiste em efetuar predições de situações desconhecidas tendo como base informação previamente conhecida (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012). Arthur Samuel (SAMUEL, 1988), primeiro a utilizar o termo Machine Learning e considerado autor do primeiro programa com capacidade de auto-aprendizado[1] (WEISS, 1992) definiu machine learning como sendo o "o campo de estudos que fornece a computadores a habilidade de aprenderem sem serem explicitamente programados".

Tipicamente subdivide-se machine learning quanto ao tipo de aprendizado, podendo ser supervisionado ou não-supervisionado. No caso de aprendizado supervisionado se tem um conjunto de dados previamente rotulados, isto é, o aprendizado se dá por exemplos. No aprendizado não-supervisionado, se possui um conjunto de dados sem nenhuma informação acerca desses dados e deseja-se encontrar padrões (HINTON, 1999).

[1] O programa é um jogo de damas, e foi descrito em seu livro *Some Studies in Machine Learning Using the Game of Checkers* (SAMUEL, 1959)

2.2 PROBLEMAS DE CLASSIFICAÇÃO

No escopo de aprendizado supervisionado existem duas categorias de problemas: os de regressão e os de classificação. A Figura 1 apresenta uma visão geral sobre como se dividem os problemas de classificação. Nos problemas de regressão obtém-se como resultado a predição de valores contínuos e no caso dos problemas de classificação, valores discretos, ou descritivos (categorias).

Em termos gerais, pode-se definir um problema de classificação como o problema de atribuir a qual categoria (de um conjunto pré-definido) uma nova observação pertence, tendo como base para a tomada de decisão um conjunto de observações cujas categorias são conhecidas de antemão. Por exemplo: dada uma mensagem de e-mail, deseja-se atribuí-la a uma categoria, onde as categorias são Spam e Não-Spam.

Um algoritmo ou técnica que implementa a classificação propriamente dita é tipicamente denominada *classificador* (SEBASTIANI, 2002).

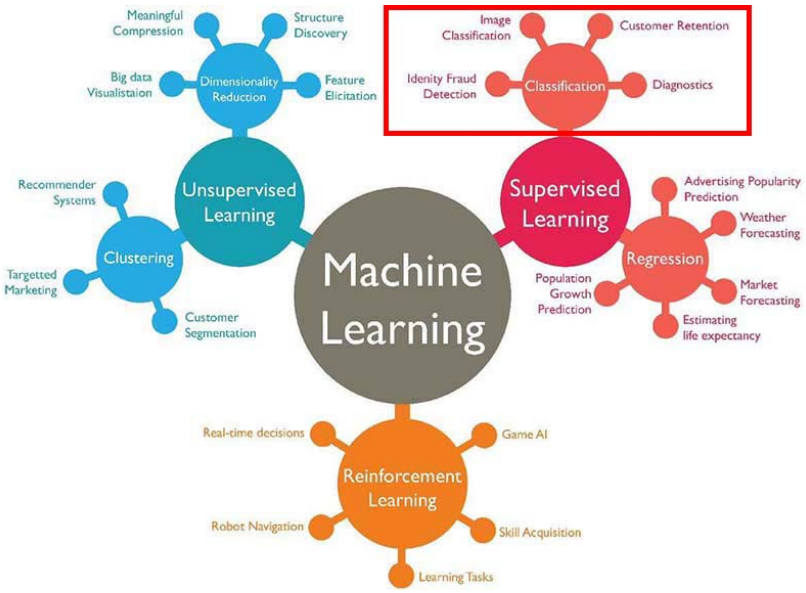

Figura 1 – Classificação dos tipos de aprendizado [2]

2.3 PROBLEMA DE CLASSIFICAÇÃO DE DOCUMENTOS

Um caso especial dos problemas de classificação é o problema de classificação de documentos, muitas vezes referenciado como *text categorization* ou *text classification* (ALLAHYARI et al., 2017) (SONG et al., 2014) (SEBASTIANI, 2002). Neste trabalho o termo utilizado será classificação de documentos, abreviado como DC, do inglês, *document classification*.

Definição 1 *No âmbito da área de classificação de documentos, um documento é uma sequência de palavras (LEOPOLD, 2002), ou texto puro.*

Exemplo 1 *São exemplos de documentos o texto contido em arquivos de texto (.txt), artigos científicos, emails, páginas web, dentre muitos outros. Note que apenas o texto contido em um arquivo é levado em consideração para classificá-lo.*

Definição 2 *Uma categoria é um rótulo simbólico, sendo que não existe nenhuma informação adicional associada a ele. Qualquer sentido ou implicação proveniente do domínio de aplicação é indiferente no âmbito do problema.*

Exemplo 2 *Um exemplo de conjunto de categorias para um problema seria Spam e Não-Spam.*

Definição 3 *Função Alvo: É uma função hipotética que prevê com 100% de acurácia todos os dados.*

Seja $D = \{d_1, d_2, ..., d_n\}$ um conjunto de documentos e $C = \{c_1, c_2, ..., c_m\}$ um conjunto de categorias pré-definidas. O problema de Classificação de Documentos consiste em atribuir um valor booleano para todo par $(d_i, c_i) \in D \times C$, sendo o valor verdadeiro quando d_i pertencer à c_i e falso caso contrário.

Deseja-se encontrar uma função $f : D \times C \rightarrow \{\text{T,F}\}$ que aproxima a *função alvo*, desconhecida $\Phi : D \times C \rightarrow \{\text{T, F}\}$. A função f é chamada de classificador ou modelo e deverá ser tal que f e Φ coincidam o máximo possível (SEBASTIANI, 2002).

[2]Extraído de https://www.techleer.com/articles/203-machine-learning-algorithm-backbone-of-emerging-technologies/

2.4 APLICAÇÕES DE CLASSIFICAÇÃO DE DOCUMENTOS

A primeira aplicação de classificação de documentos foi indexação de documentos, separando-os por assunto, descrita por Maron em 1961 (MARON, 1961). Até o presente momento DC tem sido utilizado em muitas outras aplicações, algumas das quais serão brevemente descritas nesta seção. Note que algumas aplicações não são puramente classicação de texto, mas o são pelo menos em parte.

Uma das aplicações é categorização de linguagem falada (do inglês *speech categorization*) como no trabalho desenvolvido por Myers et al. (MYERS et al., 2000). O trabalho apresenta uma maneira de classificar linguagem falada utilizando uma combinação de *speech recognition* (ou reconhecimento de fala) e DC. Sendo que primeiramente os áudios são convertidos em texto à partir de técnicas de reconhecimento de fala e posteriormente classificados utilizando técnicas de classificação de documentos.

Sable (SABLE; HATZIVASSILOGLOU, 2000) descreveu uma maneira de classificar documentos multimídia a partir da análise dos rótulos textuais dos documentos. Cavnar (CAVNAR; TRENKLE, 1994) apresentou uma aplicação de reconhecimento do idioma de textos desconhecidos. Kessler (KESSLER; NUMBERG; SCHÜTZE, 1997) apresentou identificação automático de gêneros textuais. Larkey (LARKEY, 1998) descreveu uma maneira de atribuir notas à redações e outros trabalhos textuais acadêmicos de forma automática.

2.5 CONSTRUÇÃO DE UM CLASSIFICADOR DE DOCUMENTOS

Construir um classificador de documentos não é muito diferente da construção de qualquer outro classificador ou regressor. As etapas estão ilustradas na figura 2 e serão detalhadas nas seções subsequentes.

Na figura 2 é mostrado um macro-passo nomeado de *Text Representation* cujo objetivo é produzir um *dataset* numérico a partir do conjunto de documentos de entrada, pronto para aplicação de um algoritmo de Machine Learning.

Dentro do passo de *Text Representation* é feito um pré-processamento (referenciado como *Preprocessing* na Figura 2) sobre o conjunto de documentos, pois a redução do dicionário em palavras mais significativas produz um melhor resultado (MADSEN et al., 2004). Fazem parte do pré-processamento *tokenization*, *filtering*, e *stemming* (ALLAHYARI et al., 2017). Esses passos serão descritos adiante.

Posteriormente é necessário extrair as *features*, isto é, converter o conteúdo dos documentos em variáveis numéricas para ser possível a aplicação de um algoritmo. Na seção 2.5.2 são apresentadas algumas maneiras de representação de texto em variáveis numéricas.

Uma vez tendo o conjunto de features, existe um passo denominado *feature selection*, que consiste na aplicação de métodos para reduzir a dimensionalidade do conjunto removendo *features* consideradas irrelevantes para a classificação. Além do benefício de reduzir o custo computacional dos algoritmos de classificação, este passo tende a reduzir a ocorrência de *overfitting* (IKONOMAKIS; KOTSIANTIS; TAMPAKAS, 2005).

Por fim é aplicado um algoritmo de classificação sobre o conjunto de dados pré-processado, esse algoritmo é treinado e depois validado com um conjunto de testes. Note que o pré-processamento também contribui para a redução do custo computacional do treinamento do algoritmo, pois a quantidade de dados é reduzida.

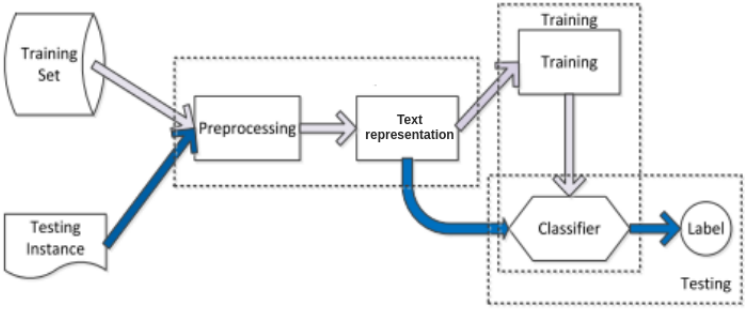

Figura 2 – Etapas da construção de um classificador de documentos [3]

2.5.1 Passos do pré-processamento

Nessa seção serão abordados os passos que compõe o macro-passo *Preprocessing*.

[3] Extraído de (SONG et al., 2014)

2.5.1.1 *Tokenization*

Este processo consiste em agrupar as sequências de caracteres em blocos denominados *tokens*. Em geral esses tokens representam palavras, termos compostos e/ou frases.

2.5.1.2 *Filtering*

Filtering é aplicado nos documentos afim de remover palavras irrelevantes. Este passo é mais comumente chamado de remoção de *stopwords*. *Stopwords* se referem à palavras comuns e repetitivas encontradas em textos escritos em linguagem natural. Dentre elas encontram-se artigos, preposições como 'para', 'de', 'até'.

A estratégia mais comum é utilizar uma lista pré-carregada de palavras, naturalmente uma lista para cada idioma, onde as palavras contidas na lista são simplesmente removidas dos documentos.

Este passo é importante pois reduz o conjunto de dados de forma significativa mantendo sua representatividade (SAIF et al., 2014).

2.5.1.3 *Feature Scaling*

Feature Scaling é um método usado para normalizar o intervalo de valores de variáveis independentes. Em processamento de dados e aprendizagem de máquina, *feature scaling* também é chamado de normalização de dados e é efetuado na etapa de pré-processamento. Essa etapa é importante para alguns algoritmos de aprendizagem de máquina, principalmente aqueles utilizados em problemas de classificação, pois acelera a sua convergência (IOFFE; SZEGEDY, 2015).

2.5.1.4 *Stemming*

Stemming, em linguística, é o processo de reduzir palavras flexionadas à sua forma base. Parte do processo consiste em reduzir verbos conjugados em tempos e pessoas diferentes ao infinitivo. Reduzir substantivos à um único gênero e grau.

Esse processo além de contribuir para reduzir a dimensionalidade do conjunto de dados aumenta a representatividade dos termos, aumentando a densidade do conjunto de dados (MADSEN et al., 2004).

A técncia de stemming mais utilizada é o algoritmo de Porter (ALLAHYARI et al., 2017) (PORTER, 1980). O exemplo da figura 3 demonstra 4 palavras sendo reduzidas à somente uma, a partir da aplicação do algoritmo de Porter. O algoritmo de Porter funciona removendo sucessivamente sufixos comuns, e é aplicável à língua inglesa.

Para a língua portuguesa temos o brasileiro RSLP (Removedor de Sufixos da Língua Portuguesa) (HUYCK; ORENGO, 2001), que funciona da mesma forma que Porter, removendo sufixos sucessivamente, porém aplicável à língua portuguesa.

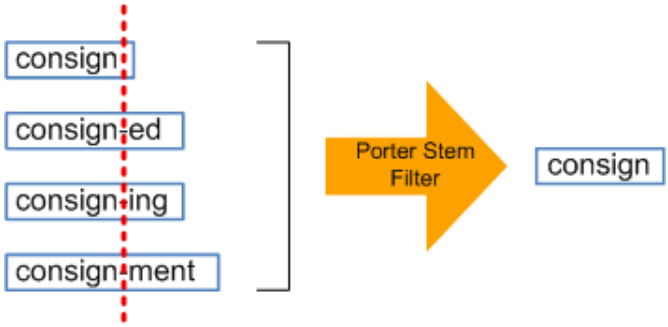

Figura 3 – Exemplo do algoritmo de Porter [4]

2.5.2 *Text Representation*

Text Representation se refere à codificação de dados textuais em variáveis numéricas. Neste trabalho será utilizado o modelo *Vector Space Model*, também referenciado como *Bag of Words*, é uma maneira de representar numericamente elementos textuais. Este modelo em sua forma mais simples representa um documento como um vetor binário, com cada posição desse vetor representando uma palavra no vocabulário (ver definição 4) do conjunto de documentos e o valor da posição representando se a palavra daquela posição pertence ao documento. Este modelo desconsidera ordenamento e gramática.

Definição 4 *Vocabulário: Conjunto de todas as palavras existentes no conjunto de documentos.*

[4] Extraído de http://xken831.pixnet.net/blog - Post Porter Stemming Algorithm

Exemplo 3 *Considere os seguintes documentos: Texto 1: "João gosta de comer maçã. Maria gosta de beber suco de maçã."*
Texto 2: "Comer maçã é bom."

Para este texto, obteremos o seguinte vocabulário V:

V = {
"João", "gosta", "de", "comer", "maçã",
"Maria", "beber", "suco", "é", "bom"
}

Considerando a definição acima, para representar os textos 1 e 2 teríamos os seguintes vetores C1 e C2, respectivamente:

$$C1 = [1, 1, 1, 1, 1, 1, 1, 1, 0, 0]$$

$$C2 = [0, 0, 0, 1, 1, 0, 0, 0, 1, 1]$$

O modelo *Bag of Words* é comumente usado em problemas de classificação para obter uma representação que possa ser utilizada como feature para treinar um classificador.

2.5.2.1 *TF-IDF*

Abreviado de *Term Frequency - Inverse Document Frequency*, TF-IDF é definido como uma estatística numérica que busca determinar quão importante uma palavra é dentro de um documento levando em consideração todo o conjunto de documentos (RAJARAMAN; ULLMAN, 2011).

No contexto do *Vector Space Model*, TF-IDF serve como um tipo de *Term Weighting*, que é uma maneira de atribuir um peso ou importância à uma dada palavra dentro de um documento. No âmbito deste trabalho cada tipo de *Term Weighting* será denominado de *Weighting Scheme*. A primeira forma de *Term Weighting* foi definida por Hans Petter Luhn (LUHN, 1957) e é resumida por: o peso de uma palavra em um documento é propocional ao *Term Frequency*, ou frequência do termo em português.

Existem diferentes maneiras de se determinar *Term Frequency*, sendo que as mais comuns são:

- Binário: Valor 0 ou 1, indicando a presença ou não da palavra no documento.
- Contagem: O número de vezes que uma palavra aparece em um documento.
- Contagem ajustada: O número de vezes que uma palavra aparece no documento divido pelo número de palavras que o documento possui. Esta é a forma que será considerada neste trabalho, pois é o esquema mais comum e atribui pesos maiores aos termos mais relevantes (ALLAHYARI et al., 2017).

Mais formalmente, neste trabalho, define-se *Term Frequency* em função da palavra e do documento.

$$tf : P \times D \rightarrow \mathbb{R} \tag{2.1}$$

Onde P é o vocabulário do conjunto de documentos e D é o conjunto de documentos.

Definição 5 *Seja a função count(p, d), cujas entradas são uma palavra p e um documento d, e seu retorno é a quantidade de vezes que p aparece em d.*

Definição 6 *Seja a função tf (Term Frequency), cujas entradas são uma palavra e um documento e seu retorno é um número real representando o peso da palavra perante àquele documento.*

$$tf(p, d) = count(p, d)/|d| \tag{2.2}$$

Note que quanto mais vezes uma palavra aparece num documento, proporcionalmente ao número total de palavras do mesmo, mais importante ela é.

Outro *Weighting Scheme* relevante para este trabalho é o *Inverse Document Frequency*, que busca representar a importância da palavra perante todos os documentos. Em termos gerais, quanto mais uma palavra aparece em outros documentos mais comum ela é e portanto menos importante.

$$idf : P \times D \times D \rightarrow \mathbb{R} \tag{2.3}$$

Onde P é o vocabulário do conjunto de documentos e D é o conjunto de documentos.

Definição 7 *Seja a função idf cujas entradas são uma palavra, um documento e o conjunto dos documentos e seu retorno é um número real representando o peso da palavra perante todos os documentos (JONES, 1972).*

$$idf(p, d, D) = \log(\frac{|D|}{\sum_{d \in D} \ count(p, d)}) \tag{2.4}$$

Por fim, tem-se o *Weighting Scheme* TF-IDF, que é a composição do *Term Frequency* com o *Inverse Document Frequency*. A importância da palavra aumenta à medida que aparece mais vezes no documento, porém ela diminui conforme aparece nos outros documentos. Isso quer dizer que palavras frequentes são importantes quando não são comuns demais. Isso evita que palavras demasiado comuns do idioma tenham importância elevada (ALLAHYARI et al., 2017).

Definição 8 *Seja a função tfidf, esta é definida em termos do produto das funções 2.2 e 2.4 apresentadas anteriormente.*

$$tfidf(p, d, D) = tf(p, d).idf(p, d, D) \tag{2.5}$$

Cerca de 70% dos sistemas de recomendação baseados em texto de trabalhos de pesquisa utiliza TF-IDF como *Weighting Scheme* (BEEL et al., 2016).

2.5.3 Treinamento

O macro-passo treinamento (referenciado na figura como *Training*) é onde será construído o classificador propriamente dito. Tendo o conjunto de dados já pré-processado, divide-se esse conjunto em dois subconjuntos, tipicamente denominados conjunto de treinamento e conjunto de teste (SEBASTIANI, 2002). O conjunto de teste será usado para avaliar a performance do classificador, processo que será detalhado na seção 2.5.4.

O processo de construção consiste em aplicar um algoritmo de ML, como *SVM*, *Random Forest*, *Naive Bayes*, dentre outros, sobre o conjunto de treinamento. Esse processo é conhecido como treinamento ou *fitting*, onde os coeficientes do algoritmo são ajustados para melhor se adequar ao conjunto de dados, sendo que o processo é diferente para cada algoritmo ou técnica. Nesta seção são apresentadas três abordagens aplicáveis à problemas de classificação.

2.5.3.1 SVM Linear

Support Vector Machine (SVM) (SCHÖLKOPF; SMOLA, 2002) é um modelo de aprendizado supervisionado aplicável à problemas de classificação. Dado um conjunto de dados previamente categorizados, o algoritmo de treinamento constrói um modelo que atribui um novo dado à uma categoria. O SVM linear é considerado um classificador linear binário, conforme será visto a seguir.

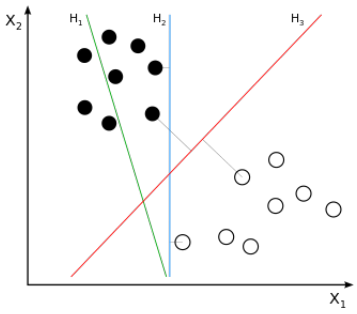

Figura 4 – Separadores [5]

Na figura 4 existem pontos de duas categorias: pontos vazios e preenchidos. Também existem as retas *H1*, *H2* e *H3*. Pode-se ver visualmente que a reta *H1* não separa os dados. As retas *H2* e *H3* ambas separam todos os dados corretamente. Porém note que a reta *H3* deixa uma "margem" maior entre os pontos. Portanto espera-se que se um novo ponto for adicionado no gráfico, será mais provável que a reta *H3* seja capaz de classificá-lo corretamente do que a reta *H2*.

Definição 9 *Hiperplano: Em geometria, um hiperplano é um subespaço cuja dimensão é uma a menos do espaço ambiente. Por exemplo, num ambiente tridimensional, um hiperplano desse espaço seria um plano. No caso de um ambiente bidimensional, um hiperplano seria uma reta (BEUTELSPACHER; ROSENBAUM, 1998).*

Essa é a ideia por trás do SVM, o algoritmo busca encontrar um hiperplano de forma a obter a maior distância dentre os pontos mais próximos de cada categoria. A figura 5 mostra um exemplo visual de um hiperplano ótimo. Nos pontos de "borda", isto é, nos pontos mais extremos das categorias, passam duas retas, uma para cada categoria.

[5] Extraído de https://en.wikipedia.org/wiki/support_vector_machine

Essas retas são denominadas vetores de suporte, e daí é proveniente o nome do algoritmo. O hiperplano é definido em termos desses vetores.

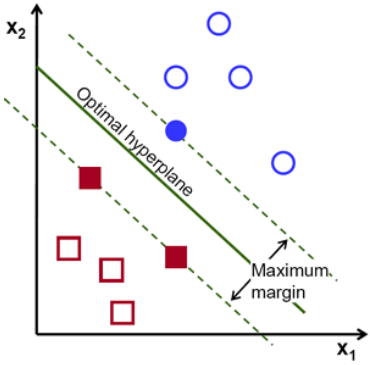

Figura 5 – Ilustração de um hiperplano ótimo [6]

A razão para a nomenclatura de hiperplano e vetores é que o algoritmo SVM é aplicável à espaços dimensionais maiores que bidimensional (BEUTELSPACHER; ROSENBAUM, 1998).

Outro aspecto interessante sobre SVM Linear é que apesar de ser um classificador linear ele é aplicável à padrões de dados que não sejam linearmente separáveis. O padrão da figura 6 claramente não pode ser separado por uma reta. Para lidar com essa situação existe um artifício matemático denominado de *kernel trick* (SCHÖLKOPF; SMOLA, 2002) que consiste em aplicar uma transformação nos dados de forma a levar os dados a um espaço dimensional superior.

Essa transformação nada mais é do que a aplicação de uma *kernel function* sobre os dados, sendo as mais populares: *Polinomial*, *Gaussiana* e *Radial Basis Function*. As figuras 6 e 7 apresentam um exemplo de um conjunto de dados onde foi aplicada a *kernel function* polinomial. Os impactos das *kernel functions* serão melhor detalhados na seção 3.3.

A figura 7 exibe os mesmos dados da figura 6 transpostos para um espaço tridimensional. Com essa transformação consegue-se separar linearmente as duas categorias a partir de um plano.

[6] Extraído de https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

[7] Extraído de https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78

[8] Extraído de https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78

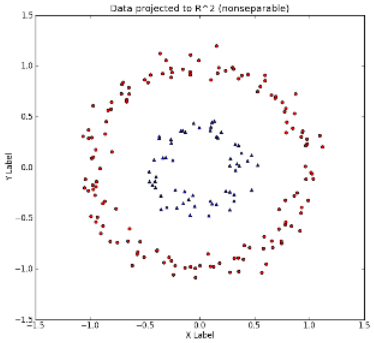

Figura 6 – Ilustração de um padrão não linearmente separável [7]

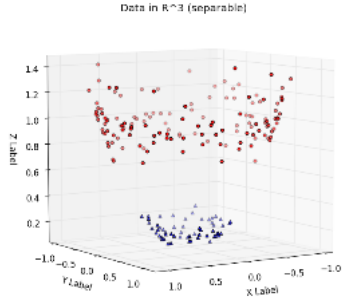

Figura 7 – Dados em um espaço dimensional superior [8]

2.5.3.2 Random Forest

O algoritmo *Random Forest* é um método de aprendizado do tipo *ensemble learning*, o qual utiliza múltiplos métodos de aprendizagem buscando obter uma performance maior do que a que seria obtida utilizando apenas um método. *Random Forest* opera construindo um conjunto de árvores de decisão na etapa de treinamento e na etapa de predição retorna como resultado final a média do resultado de múltiplas árvores (HO, 1998) (HO, 1995).

Árvores de decisão são ferramentas de aprendizado supervisionado para problemas de classificação, comumente utilizados em *data mining*. Como o próprio nome da técnica sugere, esta possui como

objetivo a construção de uma árvore que represente explicitamente a estrutura do conjunto de dados (ROKACH; MAIMON, 2005).

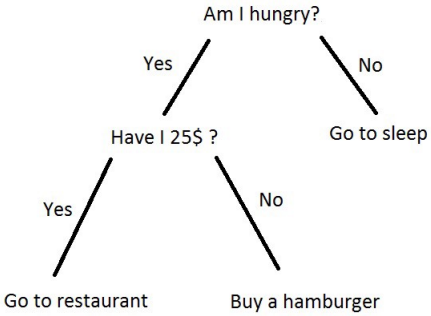

Figura 8 – Árvore de decisão representando a decisão de qual atividade executar [9]

A figura 8 mostra um exemplo de uma árvore de decisão representando a decisão de qual atividade executar. Numa árvore de decisão as folhas representam os valores alvo, ou o resultado. Enquanto os nodos internos representam as regras. No caso deste exemplo a atividade será dormir caso a pessoa não esteja com fome. Caso esteja com fome e possua mais de 25 dólares, ir à um restaurante e por fim caso esteja com fome e possua menos de 25 dólares, comprar um hambúrguer.

Em termos gerais o objetivo de uma árvore de decisão é criar um modelo que consiga predizer uma variável alvo aprendendo regras inferidas do conjunto de dados.

A figura 9 mostra visualmente a ideia de uma floresta. Cada uma das árvores é treinada com um subconjunto dos dados na etapa de treinamento. Na etapa de predição, onde se deseja obter os resultados, o algoritmo agrega os resultados através de uma função chamada *discriminador*, no caso do exempo da figura 9 é um somatório das probabilidades.

[9] Extraído de https://becominghuman.ai/understanding-decision-trees-43032111380f

[10] Extraído de https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/

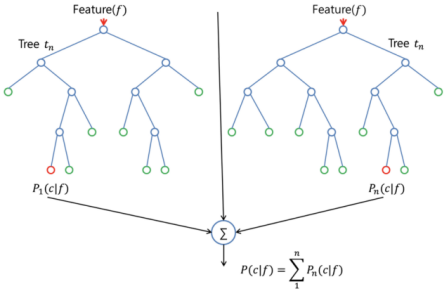

Figura 9 – Floresta [10]

2.5.3.3 Redes Neurais Artificiais

Redes neurais artificiais são um modelo matemático de aprendizagem de máquina que emula o funcionamento de uma rede de neurônios biológicos (HOPFIELD, 1982). Um neurônio biológico é uma célula do sistema nervoso, composto por dentritos, corpo celular e um axônio. Essa é uma célula capaz de receber e transmitir sinais químicos/elétricos. A maneira pela qual os neurônios se comunicam é através da conexão entre os dentritos de um neurônio com o axônio de outro. Essa conexão é denominada de sinapse, onde ocorre transmissão de sinais elétricos ou químicos via neurotransmissores.

Um único neurônio é incapaz de efetuar qualquer ação significante, porém uma rede suficientemente grande de neurônios é capaz de desempenhar funções altamente complexas, como as desempenhadas pelo cérebro humano.

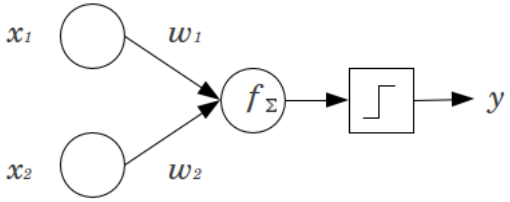

Figura 10 – Perceptron [11]

[11] Extraído de https://stats.stackexchange.com/questions/285772/what-is-weights-in-perceptron

Definição 10 *Função de ativação: função aplicada sobre o resultado de f (mostrada na figura 10) para determinar o valor de saída do neurônio (RUSSEL; NORVIG, 2004).*

O primeiro modelo matemático de uma rede neural artificial foi o *perceptron* (ROSENBLATT, 1958), ilustrado na figura 10. Conforme figura 10, o *perceptron* é composto por entradas *x1* e *x2*, saída *y* (podendo ter mais de uma saída num caso mais geral), uma função *f*, definida na equação 2.6, que aplica pesos sobre os sinais *w1* e *w2* e por fim uma função de ativação, conforme definição 10.

$$\sum_{i=1}^{m} w_i x_i \tag{2.6}$$

As funções de ativação mais comuns estão listadas abaixo, e definidas em (RUSSEL; NORVIG, 2004):

- *Threshold*. Definida na equação 2.7, possui valor 1 caso o sinal seja positivo e 0 caso seja negativo, é a função mais simples.

$$\phi(x) = \begin{cases} 1, & se\ x \geq 0 \\ 0, & se\ x < 0 \end{cases} \tag{2.7}$$

- *Sigmoid*. Definida na equação 2.8, possui valor entre 0 e 1, muito usada para predição de probabilidades.

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

- *Rectifier*. Definida na equação 2.9, possui valor entre 0 e o valor de entrada.

$$\phi(x) = max(x, 0) \tag{2.9}$$

- *Softmax*. Definida na equação 2.10, possui valor entre 0 e 1, é muito utilizada na camada de saída em problemas de classificação com múltiplas classes.

$$\phi(x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}} \tag{2.10}$$

Para construção de uma rede neural são utilizados vários neurônios. No modelo mais simples, conhecido como *multilayer perceptron*,

existe uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. O número de neurônios na camada de entrada é igual ao número de váriáveis de entrada e o número de neurônios na camada de saída é igual ao número de variáveis de saída (RUSSEL; NORVIG, 2004). A figura 11 ilustra uma rede do tipo *multilayer perceptron*, mostrando a camada de entrada em amarelo, apenas uma camada oculta em verde e a camada de saída em vermelho.

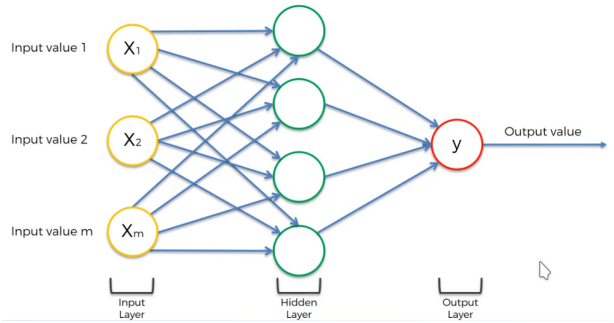

Figura 11 – Rede Neural [12]

Conforme mencionado anteriormente, cada nodo de cada camada oculta possui n entradas e n pesos associados a essas entradas, e essas entradas e pesos geram um valor de saída a partir da função 2.6. Logo, o que define o comportamento da rede como um todo, isto é, quais neurônios serão ativados ou não, são os pesos.

Para uma rede neural artificial, o processo de treinamento ou *fitting* consiste no ajuste desses pesos. Uma das maneiras mais simples de efetuar o treinamento de uma rede neural é através do algoritmo *backpropagation* (RUMELHART; HINTON; WILLIAMS, 1986). Uma vez que a rede tenha sido treinada, é possível utilizá-la para novos dados bastando aplicá-los com entrada.

2.5.4 Avaliação

Uma vez tendo construído o classificador é necessário avaliar seu desempenho ou o quão bom ele é. Nesta seção serão apresentadas algumas métricas comumente usadas para avaliação de classificadores, representando o macro-passo *Testing* da figura 2.

[12] Extraído de https://www.udemy.com/deeplearning/learn/v4/t/lecture/6747425

A avaliação de classificadores de documentos é feita de forma experimental e não analítica, pois para avaliar de forma analítica seria necessária uma especificação formal do problema; e tendo em vista que classificação de documentos possui caráter subjetivo, é inerentemente não formalizável (SEBASTIANI, 2002).

Antes de efetuar o treinamento, o conjunto de dados original é dividido em um conjunto de treinamento e um conjunto de testes, ou em inglês *test set* (HOTHO; NÜRNBERGER; PAASS, 2005). O conjunto de testes, que não é utilizado para o treinamento, é utilizado para tomar algumas medidas estatísticas. As mais comuns e utilizadas neste trabalho são *precision*, *recal*, *specificity*, *accuracy* e *f-score*. Essas métricas serão detalhadas adiante e todas são derivadas de um conceito denominado matriz de confusão.

2.5.4.1 Matriz de confusão

Em Machine Learning e mais especificamente em problemas de classificação, uma matriz de confusão (também conhecida como matriz de erros) dispõe de forma visual informações relacionadas à acurácia do modelo (STEHMAN, 1997).

É uma matriz 2x2, sendo que cada célula representa uma informação diferente acerca dos erros e acertos do modelo. Na figura 12 são mostrados as informações *true positive*, *false positive*, *true negative* e *false negative*.

- *True positive* (TP): Representa a quantidade de vezes que o modelo previu a condição como positiva e era de fato positiva. No âmbito de classificação de documentos considera-se um TP um documento que foi classificado como pertinente à uma dada categoria a qual, de fato, pertencia.

- *False positive* (FP): Representa a quantidade de vezes que o modelo previu a condição como negativa porém era positiva. No âmbito de classificação de documentos considera-se um FP um documento que foi classificado como pertinente à uma dada categoria a qual não pertencia.

- *True negative* (TN): Representa a quantidade de vezes que o modelo previu uma condição como negativa e era de fato negativa. No âmbito de classificação de documentos considera-se um TN um documento que foi classificado como não pertinente à uma dada categoria a qual de fato não pertencia.

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	True positive	False positive
	Predicted condition negative	False negative	True negative

Figura 12 – Matriz de confusão [13]

- *False negative* (FN): Representa a quantidade de vezes que o modelo previu a condição como negativa porém na verdade era positiva. No âmbito de classificação de documentos considera-se um FN um documento que foi classificado como não pertinente à uma categoria quando, de fato, pertencia.

2.5.4.2 Métricas

Nesta seção serão detalhadas as métricas utilizadas para avaliação do classificador, sendo que todas são derivadas da matriz de confusão.

Precision mede a quantidade de acertos sobre os documentos classificados como pertinentes à categoria. Pode-se entender *precision* como uma medida de quão útil o modelo é, pois ela mede, dos documentos classificados como pertinentes, quais deles realmente o eram.

$$Precision = \frac{TP}{TP + FP} \tag{2.11}$$

Recal mede a quantidade de acertos sobre os documentos pertencentes à categoria. Pode-se entender *recal* como uma medida de completude, pois mede dos documentos que deveriam ter sido classificados como pertinentes, quais de fato foram classificados como pertinentes.

$$Recal = \frac{TP}{TP + FN} \tag{2.12}$$

Accuracy é uma medida um pouco mais geral, pois mede a quantidade de acertos (positivos e negativos) sobre o total.

[13] Extraído de https://en.wikipedia.org/wiki/Confusion_matrix

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.13}$$

Por fim define-se *F-score* como sendo uma combinação de *precision* e *recal*, mais precisamente a média harmônica de ambos.

$$F1 = 2.\frac{precision.recall}{precision + recall} \tag{2.14}$$

Todas as métricas apresentadas são proporcionais, ou seja, variam de 0 à 1, sendo que quanto mais próximo de 0 pior é o resultado e quanto mais próximo de 1 melhor o resultado.

Neste capítulo foi apresentada a fundamentação teórica necessária para a compreensão do processo de construção de um classificador de documentos. No próximo capítulo será apresentada a proposta de implementação de um classificador para editais de licitação, problema original deste trabalho.

2.6 TRABALHOS RELACIONADOS

Foram elencados cinco trabalhos relacionados, todos buscados no indexador *Google Scholar*, sendo que os trabalhos elencados são provenientes das plataformas: *Semanticscholar*, *Elsevier*, *Ieeexplore* ou diretamente no site oficial do periódico. Os critérios de busca foram palavras-chave relacionadas ao domínio do problema, todas em inglês, tais como: *document classification problem*, *text classification*, *text categorization*. Foi dado foco nos trabalhos que tratavam da aplicação e avaliação das técnicas sobre conjuntos de dados conhecidos, sendo assim, de fato similares ao presente trabalho.

Manevitz et al. (MANEVITZ; YOUSEF, 2007) utiliza o algoritmo *SVM* para classificação de documentos. O conjunto de dados utilizado é o *Reuters*, um conjunto popular utilizado como *benchmark* para avaliação de técnicas de classificação de documentos. O esquema de representação utilizado foi baseado no *Bag of Words*, mais especificamente foram testadas as variações: binária, *frequency*, *TF-IDF* e *Hadamard*, sendo *weighting schemes* sobre o *Bag of Words*. Os *kernels* utilizados foram: linear, *sigmoid*, polinomial e radial. A métrica apresentada nos resultados foi a *f1-score* e obteve melhor resultado com a codificação binária e *kernel sigmoid*, atingindo 61,2% de *f1-score*.

Lung (LUNG, 2004) também utiliza o algoritmo *SVM*. O conjunto de dados é composto por notícias de um jornal chinês publicado

em todo o mundo denominado *People's Daily*. O esquema de representação utilizado foi apenas o *TF-IDF* e *kernel* polinomial. As métricas utilizadas foram *precision* e *recall*, sendo que os valores obtidos foram, respectivamente, 79,75% e 82,33%.

Klassen (KLASSEN; PATURI, 2010) utiliza o algoritmo *Random Forest*. O conjunto utilizado é um conjunto de páginas web, considerando somente o texto das páginas. O esquema de representação utilizado foi *term frequency*, onde os termos com frequência menor que cinco foram descartados. Foram feitos testes utilizando 20, 30 e 40 árvores. O experimento que obteve melhor resultado foi o realizado com 40 árvores, obtendo *precision* de 80,95%.

Manevitz (MANEVITZ; YOUSEF, 2007) utiliza uma rede neural artificial para classificação de documentos utilizando o conjunto de dados *Reuters* já mencionado. Foram feitos testes utilizando o esquema de representação *TD-IDF* e *Hadamard*. Foi utilizada uma rede *feed-forward*, contendo apenas uma camada interna composta por seis neurônios, cada um utilizando a função de ativação *sigmoid*. Este número foi escolhido de forma experimental pelos autores. Os resultados da métrica *f1-score* para a representação *TF-IDF* e *Hadamard* foram, respectivamente, 42,2% e 50,8%.

Para fins de comparação, os autores implementaram os algoritmos *Rocchio*, *Naive Bayes*, distância baseada em probabilidade, e *SVM*. De todos os algoritmos testados, o que obteve melhor resultado, isto é, métrica *f1* mais alta foi o método utilizando redes neurais. Além dessa abordagem, o autor efetuou testes com abordagens híbridas. Uma mesma entrada era aplicada em dois métodos, e numa abordagem a classificação era dita como verdadeira se ambos os métodos classificavam como verdadeiro e numa outra abordagem a classificação era dita como verdadeira se pelo menos um dos métodos classificou como verdadeiro. Porém nenhuma dessas abordagens híbridas apresentou melhora significante no resultado.

Trappey (TRAPPEY et al., 2006) também utiliza uma abordagem baseada em redes neurais. O conjunto de dados utilizado é um conjunto com registros de patentes. O esquema de representação utilizado é baseado no *bag of words*, denominado vetor de correlação originalmente proposto por (HOU; CHAN, 2003). O modelo de rede utilizado foi *Back-propagation network* com duas camadas, sendo uma camada oculta. O autor não especifica o número de neurônios na camada interna, mas afirma que o número foi escolhido de forma experimental. Na camada oculta foi utilizada a função de ativação *tansig* e na camada de saída foi utilizada a função *logsig*. O modelo obteve *precision* ligeiramente

acima de 90%.

Destes trabalhos é possível perceber que esquemas de representação baseados no *bag of words* tem sido amplamente usados com bons resultados, assim como os algoritmos *Support Vector Machine*, *Random Forest* e outros baseados em redes neurais.

3 PROPOSTA

No capítulo 2 foram apresentados os fundamentos teóricos necessários para a construção de um classificador de documentos. Conforme visto na seção 1.2, o objetivo deste trabalho é construir um classificador de editais de licitação que identifique para quais segmentos ou áreas de atuação tais editais se aplicam. As etapas da construção foram apresentadas na seção 2.5, ilustradas na figura 2.

Neste capítulo, serão detalhados todos os passos para a modelagem do problema, implementação da solução, as ferramentas e algoritmos utilizados, assim como as configurações e parâmetros associados a cada técnica. As classes, bibliotecas e especificidades das ferramentas serão apresentadas à medida que forem mencionadas.

Na seção 3.1 é apresentado o conjunto de dados que será utilizado neste trabalho, na seção 3.2 é apresentada a modelagem do problema proposto como um problema de classificação de documentos. Na seção 3.3 são apresentadas as bibliotecas e ferramentas utilizadas para implementação das técnicas já descritas na seção 2.5.3, bem como parâmetros e demais especificidades. Na subseção 3.3.1 é apresentada a implementação utilizando redes neurais artificiais, e esta possui uma subseção separada por possuir implementação diferente das demais técnicas propostas.

3.1 CONJUNTO DE DADOS

O conjunto de dados foi cedido pela empresa Liciexpress [1]. Esta empresa atua fornecendo informativos de licitação. Diariamente são coletadas informações de editais disponíveis em sites, portais governamentais e jornais. As informações dos editais são obtidas de *webcrawlers* próprios.

Obtidas essas informações, a equipe de captação de editais faz manualmente a classificação do edital em uma categoria. As categorias representam a área de atuação de uma empresa.

Exemplos de categorias:

- ADVOCACIA / JURIDICO
- AERONAVES / AVIACAO

[1] https://www.liciexpress.com.br/

- AGRICOLA / AGROPECUARIA
- AUDIO / VIDEO / SOM
- AUTOMACAO
- BRINDES / BRINQUEDOS
- CAMA / MESA / BANHO
- COMBUSTIVEL
- CONSTRUCAO CIVIL
- CONSULTORIA / ASSESSORIA
- ELETRICO / ELETRIFICACAO

Foram cedidos pela empresa aproximadamente 750 mil registros, sendo cada registro composto pelo texto descritivo do edital e pela sua categoria. Ao total existem 75 categorias distintas.

O Exemplo 4 demonstra o formato de um registro deste conjunto de dados:

Exemplo 4 *"AUTOMACAO", "AQUISICAO DE RELOGIO PONTO"*

3.1.1 Da natureza dos dados

Conforme dito anteriormente, a base de dados é real e nesta base os editais foram classificados de forma manual por operadores humanos. Em uma análise preliminar, notou-se a existência de erros de classificação que poderiam prejudicar a performance do classificador.

A fim de contornar o problema, foram buscadas maneiras de amenizar o problema identificando e removendo esses registros incorretos. Na literatura o problema de identificar esses registros incorretos ou destoantes é chamado de *outlier detection* (HODGE; AUSTIN, 2004).

Um dos métodos mais usuais para *outlier detection* é utilizar distância entre vetores. Para o problema em questão, o conjunto de documentos foi modelado como uma matriz *TF-IDF*, conforme mencionado na seção 2.5.2, onde cada linha representa um documento. Foi calculada a distância euclidiana de cada vetor representando um documento para cada vetor desta matriz de documentos, e foi calculada a média destas distâncias. O esperado seria que os documentos que estivessem mais distantes seriam *outliers*, porém observou-se que os documentos

estavam praticamente equidistantes entre si, não permitindo que fosse tirada nenhuma conclusão a partir da distância euclidiana.

Existem outras abordagens, baseadas em densidade, onde os vetores afastados de regiões densas são considerados ouliers; sendo essa abordagem similar à ideia de distância. Segundo (KANNAN et al., 2017), nenhum desses métodos é adequado para detecção de *outliers* em dados textuais em decorrência dos dados possuírem alta dimensionalidae e alta esparsidade.

No conjunto de dados existem ao total 75 categorias, das quais menos de 10 concentram mais de 80% de todos os registros. Considerou-se que as categorias que possuem poucos documentos serão agrupadas em uma categoria denominada *Outros*. A razão de agrupar em poucas categorias é reduzir o impacto dos dados incorretos sobre o desempenho, pois erros em classes com muito poucas observações irão deteriorar o desempenho como um todo.

Além disso classes com poucas observações possuem pouca relevância prática do ponto de vista do problema original, visto que o objetivo é classificar os editais visando atender segmentos de mercado. Segmentos com poucas observações significam menos oportunidades de participação em licitações, o que inerentemente significa que esses segmentos são menos relevantes.

3.2 PROBLEMA DE CLASSIFICAÇÃO DE EDITAIS DE LICITAÇÃO

O problema de classificação de editais de licitação pode ser modelado como um problema de classificação de documentos com múltiplas classes, do inglês *MultiClass Document Classification*. O problema de classificação de documentos foi definido na seção 2.3, e foi definido como um classificador binário, que sob um ponto de vista teórico pode ser considerado mais geral que o classificador multi-classe. Pois pode-se resolver o problema multi-classe utilizando exatamente os mesmos algoritmos do binário, bastando a transformação do problema multi-classe em $\{c_1, ..., c_{|C|}\}$ problemas binários, sendo $|C|$ o número de classes distintas (SEBASTIANI, 2002).

$$\{c_i, \overline{c_i}\} \forall i \in \{1, ..., |C|\} \tag{3.1}$$

Cada problema irá gerar um classificador binário para uma das classes, sendo que seu resultado será: ou pertence à classe ou não, significando no contexto geral do problema multi-classe que pertence

ou àquela classe ou à qualquer outra. Para que este método seja válido é necessário que as classes sejam estocasticamente independentes, isto é, a pertinência de um documento à uma classe não pode depender da pertinência do mesmo à nenhuma outra. Em geral essa condição é tida como verdadeira (SEBASTIANI, 2002). Esse processo será detalhado na seção 3.3.

3.3 IMPLEMENTAÇÃO

Nesta seção será apresentada a implementação do classificador propriamente dito. Analogamente à figura 2 apresentada na seção 2.5, aqui serão detalhados os passos da construção seguindo a figura 13.

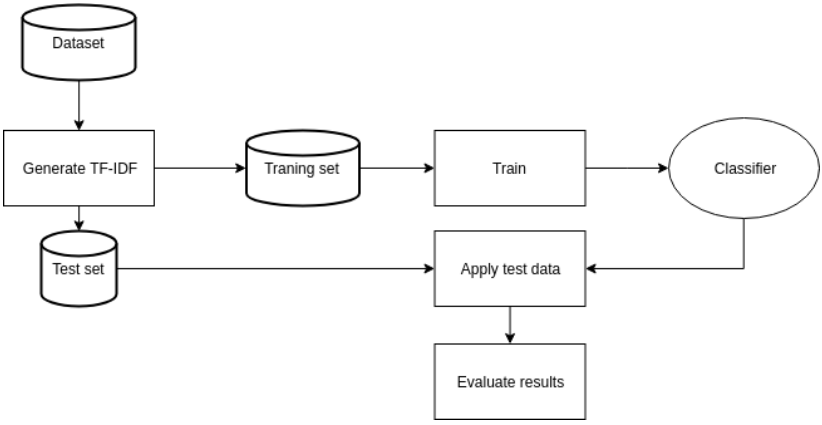

Figura 13 – Etapas de implementação [2]

- id: Número inteiro representando um identificador do documento.
- descricao: Objeto do tipo *string* representando o texto do documento.
- classe: Objeto do tipo *string* representando a classe à qual aquele documento pertence.

Após carregados os dados, para gerar o conjunto de *features* utilizando o *weighting scheme* TF-IDF, apresentado na seção 2.5.2, será utilizada a classe *TfidfVectorizer* que gera uma estrutura de dados utilizando o *TF-IDF*, a partir do conjunto de dados. Essa classe recebe

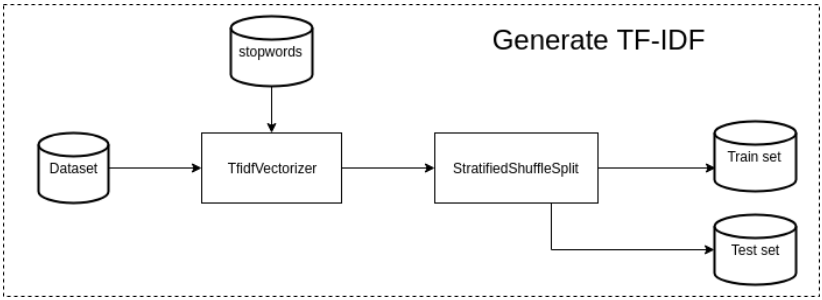

Figura 14 – Passo *Generate TF-IDF*

como parâmetro um conjunto de *stopwords* a serem removidas. O conceito de *stopwords* foi apresentado na seção 2.5.1.2. Para obtenção do conjunto de *stopword* será utilizada a classe *stopwords* da biblioteca *NLTK*[3] para processamento de linguagem natural. Essa biblioteca conta com um conjunto de *stopwords* para língua portuguesa.

Ainda sobre a classe *TfidfVectorizer*, existem dois parâmetros a serem passados.

- *max_ df*: Define o tamanho máximo da *document frequency*. Quando for construído o vocabulário para o conjunto de documentos, serão removidos os termos que tiverem *df* maior que o parâmetro *max_ df*.

- *ngram_ range*: Define os limites mínimo e máximo para construção dos *ngramas*. Os termos que comporão o esquema TF-IDF podem ser formados por termos compostos (formados por mais de uma palavra) e isto é definido a partir deste parâmetro. Definindo um limite máximo maior que um, um termo formado por mais de uma palavra seria tratado no TF-IDF como um único termo.

Para esta implementação serão testados diferentes valores para os parâmetros *max_ df* e *ngram_ range* para avaliar o impacto nos resultados. Estes parâmetros serão detalhados no capítulo 4.

O próximo passo será a divisão do conjunto de dados em dois conjuntos: o conjunto de treinamento e o conjunto de testes. Estes estão referenciados como *Training set* e *Test set* na figura 13. Para isso será utilizada a classe *StratifiedShuffleSplit* da biblioteca *Scikit Learn*.

[3] https://www.nltk.org/

Essa classe implementa amostragem aleatória e estratificada. Isso quer dizer que os conjuntos gerados a partir da amostragem possuirão a mesma proporção de documentos de cada classe do conjunto original, e a amostragem será aleatória. Isso é interessante pois classes com quantidades muito diferentes de registros deterioram o desempenho do classificador (CHAWLA et al., 2002).

Para esta classe serão passados dois parâmetros: o número de subconjuntos e o tamanho do conjunto de testes. O número de subconjuntos é 2, representando o conjunto de treinamento e o de testes. E o tamanho do conjunto de testes será 0.33, significando que o conjunto de testes terá 33% do tamanho do conjunto original.

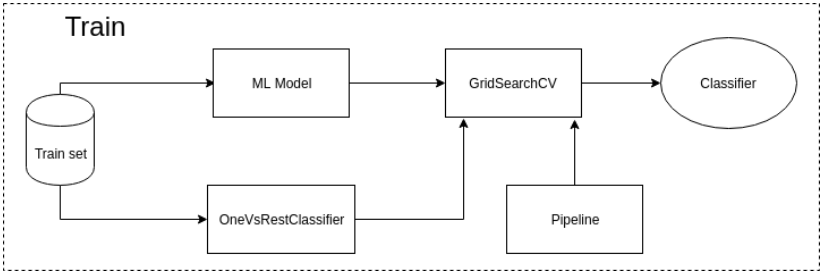

Figura 15 – Passo *Train*

Obtido o *Training set*, passa-se para o passo *Train* da figura 13, detalhado na figura 15. Neste passo será gerado o classificacador propriamente dito. Será usada a classe *OneVsRestClassifier*, cuja função é transformar o problema de classificação de documentos multi-classe em n problemas binários, conforme visto na seção 2.5. Na prática esta classe cria n classificadores. *OneVsRestClassifier* recebe como parâmetro uma classe que implementa algum algoritmo de classificação. No caso deste trabalho serão utilizadas as seguintes classes, todas da biblioteca *Scikit Learn*:

- SVC: classe que implementa um classificador utilizando o algoritmo *Support Vector Machine*, permitindo que sejam aplicadas *kernel functions* sobre o conjunto de dados. Será avaliada aqui a função *Radial Basis Function*.
- RandomForestClassifier: classe que implementa um classificador utilizando o algoritmo *Random Forest*.

SVC é uma classe referente ao algoritmo *Support Vector Machine* e *RandomForestClassifier* é referente ao algoritmo *Random Forest*; os

dois algoritmos foram apresentados na seção 2.5.3. Estes algoritmos estão referenciados como *ML Model* na figura 15. Nesse ponto tem-se n classificadores com parâmetros diferentes para *max_ df* e *ngram_ range*. Isso para cada algoritmo de treinamento. Para implementar o fitting para os algoritmos listados e obter a melhor combinação de parâmetros será utilizada a classe *GridSearchCV*. Essa classe recebe como parâmetros diferentes valores para os parâmetros *max_ df* e *ngram_ range*.

Por fim o *GridSearchCV* efetua uma busca exaustiva variando os parâmetros. *GridSearchCV* retorna um classificador com a configuração que obteve melhor resultado, permitindo aplicação do conjunto de testes para avaliação da performance do classificador. Esse processo de busca exaustiva a partir do *GridSearchCV* será repetido para todas as classes de algoritmos apresentadas nesta seção e no capítulo 4 será apresentada a forma de avaliação da performance dos modelos implementados. Ao fim do passo *Train*, será obtido o classificador, referenciado como *Classifier* na figura 13.

No passo *Apply test data* da figura 13, aplica-se o *Test set*, ou conjunto de testes, sobre o classificador já treinado, e o resultado produzido será comparado com o resultado real no passo *Evaluate results*. O passo *Evaluate results* será melhor detalhado no capítulo 4.

3.3.1 Redes Neurais Artificiais

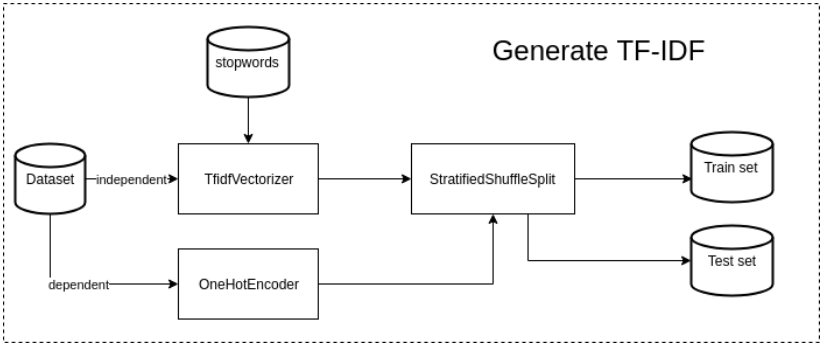

Figura 16 – Passo *Generate TF-IDF*

Além das técnicas de aprendizagem de máquina apresentadas na seção 3.3, também foi implementada uma técnica baseada em redes neurais artificiais. O framework *Scikit Learn* apesar de possuir imple-

mentação de redes neurais artificiais, esta não é a mais indicada, como afirmado pelos próprios autores (SCIKIT, 2018). Por isso optou-se por utilizar o framework *TensorFlow*[4], criado pela empresa Google e de código-aberto desde 2015. Sobre o *TensorFlow* será utilizada a API *Keras*[5] para Python.

A resolução do problema utilizando redes neurais é um pouco diferente das outras técnicas. No passo *Generate TF-IDF*, detalhado na figura 16, são geradas as *features* de forma quase idêntica à seção 3.3, porém a matriz *TF-IDF* será limitada a 20.000 colunas, selecionando aquelas que possuem maior peso. A razão dessa limitação se dá principalmente em decorrência da limitação de memória de vídeo do ambiente de treinamento, que será detalhado na seção 4.1. As variáveis que serão codificadas na matriz *TF-IDF* estão identificadas na figura 16 como *independent*, pois tratam-se das variáveis independentes.

Definição 11 *Onehot: Codificação Onehot, no âmbito de inteligência artificial, é utilizado para codificar variáveis categóricas. Nesta codificação, as variáveis codificadas possuem n dígitos, sendo n o número de classes distintas. O dígito no índice referente à classe em questão é 1 e todos os outros dígitos são 0.*

Exemplo 5 *Conjunto entrada* =

$$\{Alimento, Alimento, Veiculo, Juridico, Medicamento, Outros\}$$

Conjunto onehot =

$$\{00001, 00001, 00010, 00100, 01000, 10000\}$$

Após a geração da matriz *TF-IDF*, ainda no passo *Generate TD-IDF*, será efetuado o processo de *feature scaling*, detalhado na seção 2.5.1.3, que é basicamente uma normalização dos valores, processo que acelera a convergência do processo de aprendizado (IOFFE; SZEGEDY, 2015). Neste passo também é necessário codificar a variável dependente, referenciada na figura 16 como *dependent*, utilizando a codificação *Onehot*, conforme definição 11 e exemplo 5, de forma que se tenha um número de variáveis de saída igual ao número de classes. Para efetuar a codificação foi utilizada a classe *OneHotEncoder* da biblioteca *Scikit Learn*.

Após codificar a variável dependente o *Dataset* é dividido em *Training set* e *Test set* conforme a figura 13.

[4] https://www.tensorflow.org/?hl=pt-br
[5] https://keras.io/

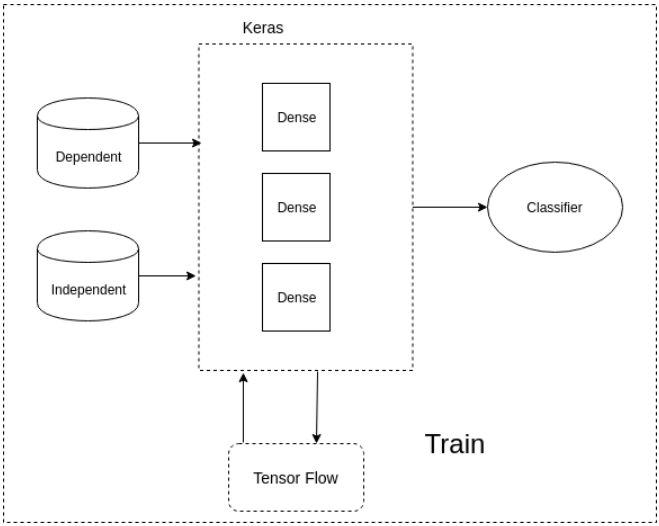

Figura 17 – Passo *Train*

No passo *Train*, detalhado na figura 17, é definida a estrutura da rede neural. A rede utilizada é do tipo *multilayer perceptron*, consistindo de 3 camadas. Para construção da rede foram utilizadas as classes *Sequencial* e *Dense* do framework *Keras*, que é uma API sobre o framework *TensorFlow* para linguagem *Python*. A classe *Sequential* define uma sequência de camadas, servindo apenas como um *wrapper* e a classe *Dense* define uma camada totalmente conectada.

As camadas internas foram construídas com $n/2$ neurônios, sendo n o número de observações. Foi utilizada a função de ativação *Rectifier*, definida na seção 2.5.3.3, e para a camada de saída foi utilizada a função *Softmax*, também definida na seção 2.5.3.3.

Diferentemente dos métodos anteriores, aqui não é necessário construir um problema binário para cada categoria, em vez disso é definida uma camada de saída com um neurônio representando cada categoria. Também por isso foi utilizada a codificação *onehot*.

No passo *Apply test data* da figura 13, aplica-se o *Test set*, ou conjunto de testes, sobre o classificador já treinado, e o resultado produzido será comparado com o resultado real no passo *Evaluate results*. O passo *Evaluate results* será melhor detalhado no capítulo 4.

4 EXPERIMENTOS E RESULTADOS

Neste capítulo será detalhado o passo *Evaluate results* da figura 13. Neste passo será avaliado o desempenho dos classificadores implementados na seção 3.3. Na seção 4.1 é detalhado o ambiente em que foram realizados os experimentos e na seção 4.2 são detalhados todos os parâmetros utilizados no treinamento dos algoritmos. Por fim na seção 4.3 é apresentado um resumo dos resultados dos algoritmos implementados na seção 3.3.

4.1 AMBIENTE DE TREINAMENTO

Para poder efetuar os experimentos referentes ao modelo de redes neurais artificiais em tempo hábil foi alocada uma máquina na AWS[1], com a seguinte especificação:

- Nome p3.2xlarge
- 1 GPU NVidia Tesla V100 com 5.120 cores
- 16 gb de memória de video
- 61 gb de memória RAM
- 8 hyper threads (considerando processador Intel Xeon E5)

A máquina foi configurada com o *tensorflow-gpu*, versão do *tensorflow* que faz uso da GPU, os devidos drivers da placa gráfica e API CUDA, que é a API que permite que programas aplicativos (nesse caso, o *tensorflow*) a acessarem instruções virtuais a serem executadas na GPU. Também foram instalados o Keras e demais bibliotecas já mencionadas. O custo da máquina *p3.2xlarge* no momento da execução dos experimentos foi de 3 dólares americanos por hora.

Para os experimentos dos algoritmos *Support Vector Machine* e *Random Forest* foi utilizada uma máquina física com a seguinte especificação:

- Nome Arucard
- 32 gb de memória RAM

[1] https://aws.amazon.com/pt/

- CPU Intel Core i7 Coffee Lake 8700
- Sem GPU

A máquina foi configurada com as bibliotecas mencionadas na seção 3.3.

4.2 PARÂMETROS E CONFIGURAÇÃO DOS EXPERIMENTOS

Para execução dos experimentos com os algoritmos *Support Vector Machine* e *Random Forest* foram selecionados os seguintes conjuntos de parâmetros para *max_df* e *ngram_range*, ambos definidos na seção 3.3. Eles foram utilizados no passo *Generate TF-IDF* conforme explicado na seção 3.3 e influenciam a maneira como a matriz *TF-IDF* é montada.

- *max_df*
$$\{0.25, 0.5, 0.75\}$$
- *ngram_range*
$$\{(1,1), (1,2), (1,3)\}$$

Já para o modelo de redes neurais artificiais foi utilizada apenas uma configuração de parâmetros, devido ao alto custo da máquina locada na AWS. No passo *Generate TF-IDF* foram utilizados os seguintes parâmetros:

- *max_features* = 20.000. Como dito na seção 3.3.1 o número de features foi limitado à 20.000.
- *max_df* = 0.75
- *ngram_range* = (1, 1)
- Número de épocas = 10

O baixo número de épocas utilizado foi definido de forma experimental, e foi baixo em função do alto custo para o treinamento utilizando a máquina *p3.2xlarge* apresentada na seção 4.1.

Dados os classificadores já treinados e o conjunto de testes, pode-se extrair métricas para avaliar a sua performance. Para isso será utilizada a classe *classification_report* da biblioteca *Scikit Learn*. Essa classe recebe como parâmetros:

- O conjunto de testes
- O resultado da aplicação do conjunto de testes sobre um dado classificador
- Os rótulos das classes dos documentos

Esta classe produz como resultado as métricas de *precision*, *recall* e *f1-score* individualmente para cada classe e também a média geral.

4.3 RESULTADOS

Nessa seção serão apresentados os resultados de desempenho de todos os algoritmos implementados na seção 3.3. As métricas apresentadas são *Precision*, *Recall* e *f1-score*. Todas essas métricas já foram descritas na seção 2.5.4, e os relatórios das tabelas 1, 2 e 3 foram gerados a partir da classe *classification_report* apresentada na seção 4.2.

4.3.1 *Random Forest*

Com o algoritmo *Random Forest*, se obteve *precision* média de 79%, *recall* médio de 79% e *f1-score* médio de 79%, ficando, portanto, abaixo do mínimo desejável de 80% para as métricas mencionadas e ainda abaixo que a implementação com *SVM*.

A figura 18 apresenta a matriz de confusão resultante da aplicação do algoritmo *Random Forest*. Esta é uma matriz normalizada, portanto seus valores estão em percentual. Na diagonal principal encontram-se os valores de verdadeiro positivo para cada classe. Nas demais células encontram os erros de classificação.

É possível observar que os erros se concentram na classe 'Outros'. Conhecendo o domínio do problema e sabendo-se que existem muitos erros de classificação nos dados de entrada, o que se pode inferir é que de fato os erros de classificação do conjunto de entrada (note que o conjunto de entrada foi classificado manualmente) tiveram forte influência sobre o resultado.

Em especial notam-se as classes 'Informática' e 'Médico / Hospitalar'. Nessas classes existem muitos termos técnicos muito específicos e incomuns , como criocautério e colposcópio, que deveriam ser classificados como 'Médico / Hospitalar'; porém muitas vezes são classificados de forma diferente.

Tabela 1 – Resultados *Random Forest*

	Precision	Recall	F1-score
COMBUSTIVEL	0.94	0.95	0.94
CONSTRUCAO CIVIL	0.78	0.62	0.69
ELETRICO / ELETRIFICACAO	0.86	0.70	0.77
ENGENHARIA / ARQUITETURA	0.71	0.66	0.69
ESCRITORIO / EXPEDIENTE	0.87	0.80	0.84
GENEROS ALIMENTICIO / ALIMENTO	0.87	0.76	0.81
INFORMATICA	0.78	0.57	0.66
LIMPEZA / HIGIENE	0.78	0.69	0.73
MAQUINA / EQUIPAMENTO PESADO	0.73	0.64	0.68
MEDICAMENTO / FARMACIA	0.87	0.78	0.82
MEDICO / HOSPITALAR	0.71	0.50	0.59
Outros	0.77	0.91	0.83
VEICULO	0.76	0.76	0.76
Média	0.79	0.79	0.79

Os exemplos a seguir mostram alguns exemplos de classificações corretas e incorretas demonstrando erros envolvendo a classe 'Outros':

Exemplo a)
Texto: AQUISICAO DE MEDICAMENTOS
Resultado estimado: MEDICAMENTO / FARMACIA
Resultado esperado: MEDICAMENTO / FARMACIA

Exemplo b)
Texto: REGISTRO DE PRECO PARA AQUISICAO DE FLORES DE EPOCA, ARBUSTOS ORNAMENTAIS, PEDRAS ORNAMENTAIS, GRAMA, ADUBO E CALCARIO EM ATENDIMENTO A SECRETARIA MUNICIPAL DA CIDADE
Resultado estimado: Outros
Resultado esperado: Outros

Exemplo c)
Texto: CONTRATACAO DE EMPRESA PARA FORNECIMENTO DE PECAS E MAO DE OBRA MECANICA PARA A FROTA DO MUNICIPIODE BARBOSA FERRAZ - PR
Resultado estimado: VEICULO
Resultado esperado: VEICULO

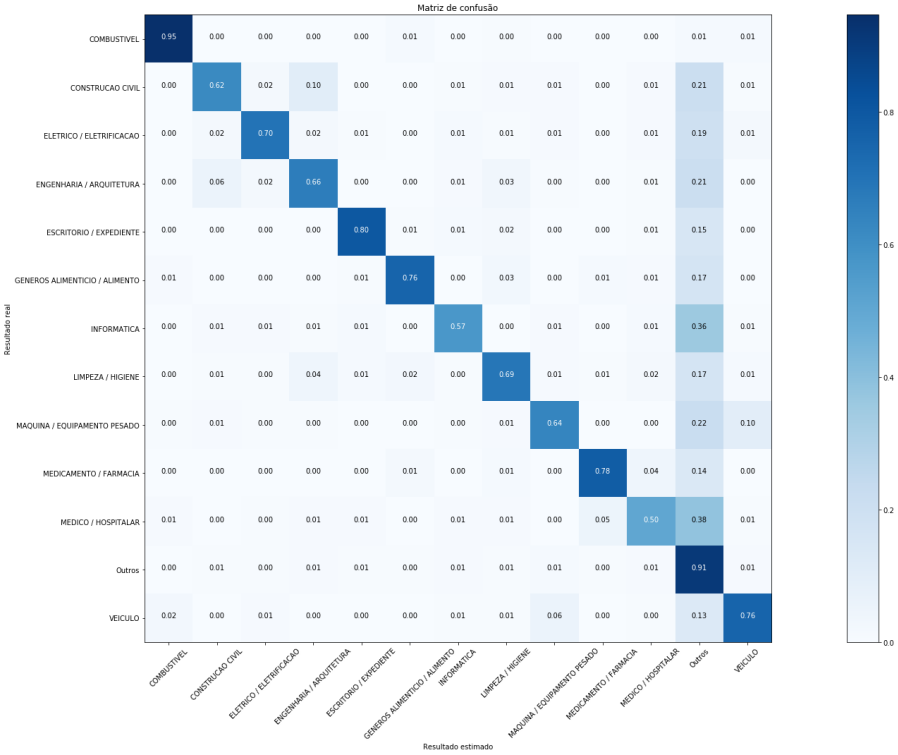

Figura 18 – Matriz de confusão *Random Forest*

Exemplo d)
Texto: CONTRATACAO DE EMPRESA PARA MANUTENCAO DE ESPECTROFOTOMETRO, ESTUFA INCUBADORA E ULTRAFREEZER
Resultado estimado: Outros
Resultado esperado: MEDICO / HOSPITALAR

Exemplo e)
Texto: CONTRATACAO DE EMPRESA ESPECIALIZADA PARA FORNECIMENTO, POR MEIO DE REGISTRO DE PRECOS, DE CARTUCHOS DE TINTA E TONERS, PARA ATENDIMENTO DAS DEMANDAS DAS SECRETARIAS MUNICIPAIS DO MUNICIPIO DE COLOMBO, CONFORME QUANTIDADES E ESPECIFICACOES CONSTANTES NO TERMO DE REFERENCIA (ANEXO VII), QUE

INTEGRA O EDITAL.
Resultado estimado: Outros
Resultado esperado: INFORMATICA

4.3.2 *Support Vector Machine*

Tabela 2 – Resultados *SVM*

	Precision	Recall	F1-score
COMBUSTIVEL	0.88	0.94	0.91
CONSTRUCAO CIVIL	0.75	0.67	0.71
ELETRICO / ELETRIFICACAO	0.83	0.68	0.75
ENGENHARIA / ARQUITETURA	0.76	0.37	0.50
ESCRITORIO / EXPEDIENTE	0.85	0.77	0.81
GENEROS ALIMENTICIO / ALIMENTO	0.79	0.83	0.81
INFORMATICA	0.87	0.73	0.79
LIMPEZA / HIGIENE	0.77	0.72	0.74
MAQUINA / EQUIPAMENTO PESADO	0.82	0.59	0.69
MEDICAMENTO / FARMACIA	0.87	0.85	0.86
MEDICO / HOSPITALAR	0.84	0.52	0.64
Outros	0.68	0.89	0.7
VEICULO	0.79	0.76	0.78
Média	0.77	0.76	0.76

Com o algoritmo *SVM*, se obteve *precision* média de 77%, *recall* médio de 76% e *f1-score* médio de 76%, ficando, portanto, abaixo do mínimo desejável de 80% para as métricas mencionadas.

A figura 19 apresenta a matriz de confusão resultante da aplicação do algoritmo *SVM*. De forma análoga à do algoritmo da subseção 4.3.1, o resultado foi muito semelhante. A classe que concentrou a maior parte dos erros foi a classe 'Outros'.

Os exemplos a seguir mostram alguns exemplos de classificações corretas e incorretas demonstrando erros envolvendo a classe 'Outros':

Exemplo a)
Texto: CONTRATACAO DE EMPRESA ESPECIALIZADA EM PRESTACAO DE SERVICOS DE CONSULTORIA E CONFECCAO DE CONJUNTO DE DOCUMENTOS (PROGRAMA DE PREVENCAO DE RISCOS AMBIENTAIS E LAUDOS TECNICOS DAS CONDI-

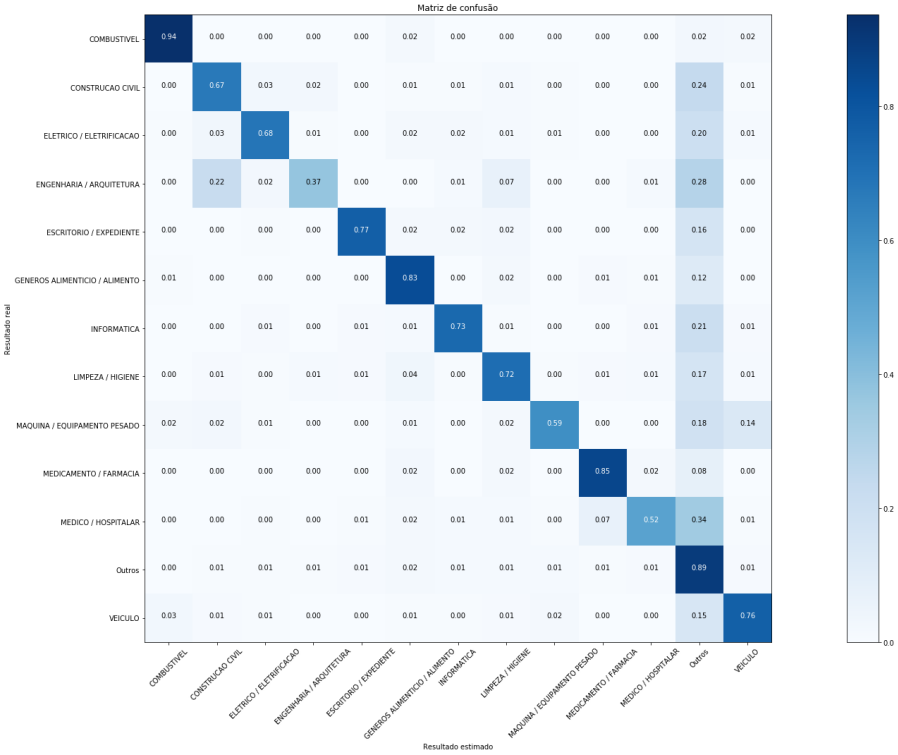

Figura 19 – Matriz de confusão *SVM*

COES AMBIENTAIS DO AMBIENTE DE TRABALHO)
Resultado estimado: Outros
Resultado esperado: Outros

Exemplo b)
Texto: AQUISICAO DE GAS DE COZINHA EM BOTIJAO DE 13KG E AGUA MINERAL EM GALAO COM 20 LITROS, PARA ATENDER A DEMANDA DOS DEPARTAMENTOS DA PREFEITURA MUNICIPAL DE BOA ESPERANCA DO IGUACU/PR
Resultado estimado: GENEROS ALIMENTICIO / ALIMENTO
Resultado esperado: COMBUSTIVEL

Exemplo c)
Texto: AQUISICAO DE PECAS PARA KOMBI, PLACAS IRL 1393,

PARA A SECRETARIA DE MEIO AMBIENTE E PLANEJAMENTO.
Resultado estimado: Outros
Resultado esperado: VEICULO

Exemplo d)
Texto: AQUISICOES DE COMBUSTIVEIS PARA CONSUMO DA FROTA DA PREFEITURA MUNICIPAL, FUNDO MUNICIPAL DE SAUDE, FUNDO MUNICIPAL DE AGRICULTURA, MEIO AMBIENTE, CORPO DE BOMBEIROS E POLICIA MILITAR, DO MUNICIPIO DE FORQUILHINHA, DURANTE O EXERCICIO DE 2018.
Resultado estimado: COMBUSTIVEL
Resultado esperado: COMBUSTIVEL

Exemplo e)
Texto: PRESTACAO DE SERVICO EXPECIALIZADO EM TRANSBORDO E DESTINADO FINAL DE RESIDUOS SOLIDOS URBANOS
Resultado estimado: LIMPEZA / HIGIENE
Resultado esperado: ENGENHARIA / ARQUITETURA

4.3.3 Redes Neurais

Utilizando *Redes Neurais*, se obteve *precision* média de 88%, *recall* médio de 88% e *f1-score* médio de 88%, atingindo o mínimo desejável.

Como pode-se observar nas tabelas 1, 2 e 3, a técnica que obteve o melhor desempenho foi a técnica baseada em Redes Neurais Artificiais, obtendo *Precision*, *Recall*, *f1-score* iguais a 88%. E esta foi a única técnica que obteve êxito em atingir o mínimo desejável de 80% em todas as métricas mencionadas. A matriz de confusão não foi gerada para esta técnica, porém é esperado que seu formato seja similar às matrizes das outras técnicas, que apresentaram os erros todos concentrados na classe 'Outros'.

Tabela 3 – Resultados *ANN*

	Precision	Recall	F1-score
COMBUSTIVEL	0.95	0.91	0.93
CONSTRUCAO CIVIL	0.89	0.93	0.91
ELETRICO / ELETRIFICACAO	0.88	0.75	0.81
ENGENHARIA / ARQUITETURA	0.72	0.65	0.69
ESCRITORIO / EXPEDIENTE	0.90	0.82	0.86
GENEROS ALIMENTICIO / ALIMENTO	0.95	0.95	0.95
INFORMATICA	0.89	0.91	0.90
LIMPEZA / HIGIENE	0.84	0.85	0.84
MAQUINA / EQUIPAMENTO PESADO	0.72	0.83	0.77
MEDICAMENTO / FARMACIA	0.87	0.89	0.88
MEDICO / HOSPITALAR	0.87	0.89	0.88
Outros	0.88	0.88	0.88
VEICULO	0.93	0.92	0.92
Média	0.88	0.88	0.88

5 CONCLUSÃO

Este trabalho possibilitou verificar a viabilidade de automatizar o processo de classificação dos editais licitatórios em empresas de informativos de licitação utilizando técnicas de aprendizagem de máquina.

Demonstrou-se uma modelagem do problema como um problema de classificação de documentos, da área de aprendizado supervisionado. Foram apresentadas técnicas aplicáveis ao problema de classificação de documentos e maneiras para avaliar o desempenho das técnicas aplicadas. As técnicas foram aplicadas sobre uma base de dados real, portanto com erros de classificação. Foram avaliadas abordagens para remover os erros e por fim optou-se por adaptar o conjunto de dados a fim de tolerar os erros.

Dentro do objetivo deste trabalho de construir um classificador de documentos capaz de classificar de forma satisfatória os editais de licitação, onde havia sido estipulado que as métricas de desempenho teriam que ter valor mínimo de 80%, obteve-se um valor de 88%, atingindo assim, o objetivo do trabalho.

O classificador obtido pode fazer parte do processo automático de captação de editais já existente em empresas de informativos de licitação, como é o caso da empresa Liciexpress, cujo processo foi estudado para elaboração deste trabalho. Dessa forma substituindo classificação manual e reduzindo custos operacionais.

5.1 TRABALHOS FUTUROS

Na seção 3.1 foi dito que haviam erros nos dados e foram apresentadas algumas abordagens para detecção de *outliers*, porém nenhuma obteve um bom resultado. Foi tomado conhecimento da abordagem baseada em *Non-negative Matrix Factorization*, que apresenta bons resultados aplicada à texto (KONG; DING; HUANG, 2011) (KANNAN et al., 2017), porém no momento em que foi tomado conhecimento sobre esta abordagem não havia tempo hábil para incorporação desta ao trabalho, ficando assim, a ser incluído no futuro.

Na seção 4.2 foi apresentada a única configuração de parâmetros para a implementação baseada em redes neurais. Foi utilizada apenas uma configuração com um baixo número de épocas devido ao alto custo de treinamento, ficando para um outro momento avaliar o mesmo modelo utilizando diferentes combinações dos parâmetros mostrados com

um número maior de épocas. Além disso também podem ser explorados outros modelos de redes neurais, como Redes Neurais Convolucionais (PARIKH et al., 2016) e Redes Neurais Convolucionais Recorrentes (LAI et al., 2015). Além disso seria interessante efetuar uma avaliação mais detalhada das saídas da rede neural. Aqui foi utilizada a função *softmax*; poderia-se analisar a diferença entre as saídas para ver o quão distante ficaram.

Também é válido avaliar a introdução de heurísticas, tais como uma lista de palavras previamente selecionadas, gerada a partir de conhecimento sobre o domínio do problema. Essas palavras seriam parte do conjunto de *features* da matriz *TF-IDF*.

REFERÊNCIAS

ALLAHYARI, M. et al. A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques. *Proceedings of KDD Bigdas*, p. 1—-13, 2017.

BEEL, J. et al. Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, Springer Berlin Heidelberg, v. 17, n. 4, p. 305–338, nov 2016. ISSN 1432-5012.

BEUTELSPACHER, A.; ROSENBAUM, U. *Projective Geometry: From Foundations to Applications*. [S.l.: s.n.], 1998. ISBN 9780521483643.

CAVNAR, W. B.; TRENKLE, J. M. N-Gram-Based Text Categorization. In: *In Proceedings of the 3º Annual Symposium on Document Analysis and Information Retrieval*. [S.l.: s.n.], 1994.

CHAWLA, N. V. et al. *SMOTE: Synthetic Minority Over-sampling Technique*. [S.l.], 2002. v. 16, 321–357 p. <https://arxiv.org/pdf/1106.1813.pdf>.

HINTON, T. J. S. G. *Unsupervised Learning: Foundations of Neural Computation (Computational Neuroscience)*. [S.l.]: A Bradford Book, 1999. ISBN 026258168X.

HO, T. K. Random decision forests. In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. [S.l.]: IEEE Comput. Soc. Press, 1995. v. 1, p. 278–282. ISBN 0-8186-7128-9. ISSN 0818671289.

HO, T. K. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 20, n. 8, p. 832–844, 1998. ISSN 01628828.

HODGE, V. J.; AUSTIN, J. A Survey of Outlier Detection Methodoligies. *Artificial Intelligence Review*, v. 22, n. 1969, p. 85–126, 2004.

HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities (associative memory/parallel processing/categorization/content-addressable memory/fail-soft devices). v. 79, p. 2554–2558, 1982.

HOTHO, A.; NÜRNBERGER, A.; PAASS, G. A Brief Survey of Text Mining. *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, v. 20, p. 19–62, 2005. ISSN 01751336.

HOU, J.-L.; CHAN, C.-A. a Document Content Extraction Model Using Keyword Correlation Analysis. *International Journal of Electronic Business Management*, v. 1, n. 1, p. 54–62, 2003. <https://pdfs.semanticscholar.org/3acf/291fa2476132916af5b270b8ef49a40a2821.pdf>.

HUYCK, C.; ORENGO, V. A stemming algorithmm for the portuguese language. In: *String Processing and Information Retrieval, International Symposium on(SPIRE)*. [s.n.], 2001. v. 00, p. 0186. <doi.ieeecomputersociety.org/10.1109/SPIRE.2001.10024>.

IKONOMAKIS, M.; KOTSIANTIS, S.; TAMPAKAS, V. Text classification using machine learning techniques. *WSEAS Transactions on Computers*, v. 4, n. 8, p. 966–974, 2005. ISSN 11092750.

IOFFE, S.; SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. ISSN 0717-6163. <https://arxiv.org/pdf/1502.03167.pdf http://arxiv.org/abs/1502.03167>.

JONES, K. S. A Statistical Interpretation of Term Specificity and its Retrieval. *Journal of Documentation*, v. 28, n. 1, p. 11–21, 1972. ISSN 0022-0418.

KANNAN, R. et al. Outlier Detection for Text Data : An Extended Version. In: *SIAM Data Mining Conference*. [S.l.: s.n.], 2017. ISBN 9781611974874.

KESSLER, B.; NUMBERG, G.; SCHÜTZE, H. Automatic detection of text genre. In: *Proceedings of the 35th annual meeting on Association for Computational Linguistics* -. [S.l.: s.n.], 1997. p. 32–38. ISSN 1651-226X.

KLASSEN, M.; PATURI, N. Web document classification by keywords using random forests. In: *Communications in Computer and Information Science*. [s.n.], 2010. v. 88 CCIS, n. PART 2, p. 256–261. ISBN 3642143059. ISSN 18650929. <https://pdfs.semanticscholar.org/b489/157e3129d2b9a38a76c659a765be3ab8b2bc.pdf

KONG, D.; DING, C.; HUANG, H. Robust nonnegative matrix factorization using l21-norm. In: *Proceedings of the 20th ACM*

International Conference on Information and Knowledge Management. [S.l.: s.n.], 2011. (CIKM '11), p. 673–682. ISBN 978-1-4503-0717-8.

LAI, S. et al. Recurrent Convolutional Neural Networks for Text Classification. *Twenty-Ninth AAAI Conference on Artificial Intelligence*, p. 2267–2273, 2015.

LARKEY, L. S. Automatic essay grading using text categorization techniques. *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '98*, n. 1, p. 90–95, 1998. ISSN 01635840.

LEOPOLD, E. Text Categorization with Support Vector Machines. How to Represent Texts in Input Space? *Machine Learning*, v. 46, p. 423–444, 2002.

LUHN, H. P. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development*, v. 1, n. 4, p. 309–317, 1957. ISSN 0018-8646.

LUNG, J.-z. SVM multi-classifier and Web document classification. *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, n. August, p. 1347–1351, 2004. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1381982>.

MADSEN, R. et al. Pruning the vocabulary for better context recognition. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* [S.l.]: IEEE, 2004. p. 483–488. ISBN 0-7695-2128-2.

MANEVITZ, L.; YOUSEF, M. One-class document classification via Neural Networks. *Neurocomputing*, v. 70, n. 7-9, p. 1466–1481, 2007. ISSN 09252312.

MARON, M. E. Automatic Indexing: An Experimental Inquiry. *Journal of the ACM*, v. 8, n. 3, p. 404–417, 1961. ISSN 00045411.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. *Foundations of machine learning*. [S.l.]: MIT Press, 2012. 414 p. ISBN 9780262018258.

MYERS, K. et al. A Boosting Approach to Topic Spotting on Subdialogues. *Proceedings of ICML-00, 17th International Conference on Machine Learning*, n. June 2014, p. 655–662, 2000.

PARIKH, A. P. et al. Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks. *NAACL*, 2016. ISSN 0001-0782.

PORTER, M. F. An Algorithm for Suffix Stripping. *Program: Electronic Library and Information Systems*, v. 14, 1980.

RAJARAMAN, A.; ULLMAN, J. D. Data Mining. *Mining of Massive Datasets*, v. 18 Suppl, p. 114–142, 2011. ISSN 00401706.

ROKACH, L.; MAIMON, O. Top-down induction of decision trees classifiers - A survey. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, v. 35, n. 4, p. 476–487, 2005. ISSN 10946977.

ROSENBLATT, F. *THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN 1*. 1958. 19–27 p.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, v. 323, n. 6088, p. 533–536, 1986.

RUSSEL, S.; NORVIG, P. *Artificial Intelligence A Modern Approach (3rd Edition)*. [S.l.: s.n.], 2004. ISBN 9780136042594.

SABLE, C. L.; HATZIVASSILOGLOU, V. Text-based approaches for non-topical image categorization. *International Journal on Digital Libraries*, v. 3, n. 3, p. 261–275, 2000. ISSN 14325012.

SAIF, H. et al. On Stopwords, Filtering and Data Sparsity for Sentiment Analysis of Twitter. *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, n. i, p. 810–817, 2014.

SAMUEL, A. L. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, v. 3, n. 3, p. 210–229, 1959. ISSN 0018-8646.

SAMUEL, A. L. Some Studies in Machine Learning Using the Game of Checkers. I. In: *Computer Games I*. New York, NY: Springer New York, 1988. p. 335–365. ISBN 9781461387183.

SCHÖLKOPF, B.; SMOLA, A. J. Support Vector Machines and Kernel Algorithms. *The Handbook of Brain Theory and Neural Networks*, p. 1119–1125, 2002.

SCIKIT. *Scikit Learn, Neural Networks Models*. 2018. http://scikit-learn.org/stable/modules/neural_networks_supervised.html. Accessed: 2018-10-01.

SEBASTIANI, F. Machine learning in automated text categorization. *Csur*, v. 34, n. 1, p. 1–47, 2002.

SONG, G. et al. Short Text Classification: A Survey. *Journal of Multimedia*, v. 9, n. 5, p. 635–643, 2014. ISSN 1796-2048.

STEHMAN, S. V. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, v. 62, n. 1, p. 77–89, 1997. ISSN 00344257.

TRAPPEY, A. J. et al. Development of a patent document classification and search platform using a back-propagation network. *Expert Systems with Applications*, v. 31, n. 4, p. 755–765, 2006. ISSN 09574174. <www.elsevier.com/locate/eswa>.

WEISS, E. Biographies: Eloge: Arthur Lee Samuel (1901-90). *IEEE Annals of the History of Computing*, v. 14, n. 3, p. 55–69, 1992. ISSN 1058-6180.