



**Fábio Corrêa Cordeiro**

**Petrolês - Construção de um corpus  
especializado em óleo e gás em português**

Monografia apresentada ao Departamento de Engenharia Elétrica da PUC-Rio como requisito parcial para a obtenção do título de Especialização em Business Intelligence.

Orientador: Prof. Cristian Muñoz

Rio de Janeiro  
Janeiro de 2020



**Fábio Corrêa Cordeiro**

**Petrolês - Construção de um corpus  
especializado em óleo e gás em português**

Monografia apresentada ao Departamento de Engenharia Elétrica da PUC-Rio como requisito parcial para a obtenção do título de Especialização em Business Intelligence. Aprovada pela Comissão Examinadora abaixo.

**Prof. Cristian Muñoz**

Orientador

Departamento de Engenharia Elétrica – PUC-Rio

Rio de Janeiro, 20 de janeiro de 2020

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

### **Fábio Corrêa Cordeiro**

Engenheiro de produção graduado na Universidade Federal do Rio de Janeiro (UFRJ) e pós-graduado em Gerenciamento Estratégico da Inovação Tecnológica na Universidade Estadual de Campinas (UNICAMP). Desde 2007, trabalha no Centro de Pesquisas da Petrobras (CENPES) em diversos projetos de P&D, interagindo com as principais instituições de pesquisa brasileiras. Atualmente, está envolvido com projetos de tecnologias digitais, principalmente nas áreas de busca semântica e processamento de linguagem natural.

#### Ficha Catalográfica

Corrêa Cordeiro, Fábio

Petrolês - Construção de um corpus especializado em óleo e gás em português / Fábio Corrêa Cordeiro; orientador: Cristian Muñoz. – Rio de Janeiro: PUC-Rio, Departamento de Engenharia Elétrica, 2020.

v., 87 f: il. color. ; 30 cm

Monografia - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica.

Inclui bibliografia

1. Processamento de Linguagem Natural;. 2. Corpus;. 3. Óleo & Gás.. I. Muñoz, Cristian. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. III. Título.

CDD:

## Resumo

Corrêa Cordeiro, Fábio; Muñoz, Cristian. **Petrolês - Construção de um corpus especializado em óleo e gás em português**. Rio de Janeiro, 2020. 87p. Monografia – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Algoritmos baseados em aprendizado de máquina são, por definição, dependentes de um conjunto de dados para a realização do seu treinamento. Não é diferente com os algoritmos de processamento de linguagem natural (PLN). No entanto, há uma grande dificuldade para o desenvolvimento de modelos de PLN pela indústria do petróleo no Brasil. Há uma carência de bases de documentos de referência que sirvam como conjunto de dados para a realização desses treinamentos. Portanto, o objetivo desse trabalho foi criar o Petrolês, um corpus público formado por teses e dissertações em português, no domínio de óleo e gás. O Petrolês será referência para os grupos de pesquisas em inteligência artificial e empresas relacionadas à indústria do petróleo.

## Palavras-chave

Processamento de Linguagem Natural; Corpus; Óleo & Gás.

## Abstract

Corrêa Cordeiro, Fábio; Muñoz, Cristian (Advisor). **Petrolês - A specialized oil and gas corpus in Portuguese**. Rio de Janeiro, 2020. 87p. Monograph – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Machine learning algorithms are, by definition, based on a dataset for their training. It is not different for the development of Natural Language Processing (NLP) models. However, there is great difficulty in the development of NLP models by the oil and gas industry in Brazil. There is a lack of reference documents dataset that can be used to train the models. Therefore, the main aim of this work was to build the Petrolês, a public corpus compiled by thesis and dissertations in Portuguese, in the oil and gas domain. Petrolês will become a reference for artificial intelligence research groups and companies from the oil industry.

## Keywords

Natural Language Processing; Corpus; Oil & Gas.

## **Sumário**

1	Introdução	13
2	Corpus	14
3	Teses e Dissertações em Óleo e Gás	19
4	Classificação de Textos	25
5	Metodologia	28
5.1	Crawler da BDTD	28
5.2	Classificador de textos	31
5.3	Teses relevantes para o domínio óleo e gás	36
5.4	Crawler para os repositórios institucionais	37
5.5	Extração de informações dos documentos	39
6	Conclusão	41
7	Apêndice 1 - Notebook webscraping BDTD	45
8	Apêndice 2 - Notebook classificação de texto	49
9	Apêndice 3 - Notebook webscrapping repositórios institucionais	80

## Lista de figuras

Figura 3.1	Quantidade de documento por instituição [11]	20
Figura 3.2	Instituições Credenciadas pela ANP até 30 de setembro 2019 [3]	21
Figura 3.3	Páginas iniciais e finais: Capa, agradecimentos, sumários e bibliografia [4]	22
Figura 3.4	Conteúdo dos capítulos: Texto, figuras, tabelas e fórmulas [4]	23
Figura 5.1	Etapas para a criação de um corpus de textos no domínio de óleo e gás em português. (Fonte: Elaborada pelo autor)	28
Figura 5.2	Exemplo de metadados no <i>website</i> da BDTD. [20]	29
Figura 5.3	Exemplo do código HTML na BDTD. [20]	30
Figura 5.4	Representação de textos usando vetorização de palavras. [14]	34
Figura 5.5	Exemplo de metadados de uma dissertação gravado em formato JSON. (Fonte: Elaborada pelo autor)	38
Figura 5.6	Identificando <i>tag</i> para recuperar documento no repositório institucional da UFBA[7]	38

## Lista de tabelas

Tabela 2.1	Corpora mais comuns[2]	17
Tabela 2.2	Corpora mais comuns em português [17]	17
Tabela 5.1	Acurácia dos modelos de classificação	36
Tabela 5.2	Documentos recuperados por instituição	37



## Lista de Códigos

Código 1	Função para coletar metadados de um documento na BDTD	30
Código 2	Criando os vetores de frequência de palavras e os vetores TF-IDF	33
Código 3	Treinando e indexando os textos com o modelo Word2Vec	34
Código 4	Função para treinar modelo de classificação de textos	35
Código 5	<i>Crawler</i> para recuperar todas os documentos do repositório institucional da UFBA	38

## Lista de Abreviaturas

ANP – Agência Nacional de Petróleo, Gás Natural e Biocombustível

BDTD – Base Digital de Teses e Dissertação

BERT – *Bidirectional Encoder Representations from Transformers*

CNPQ – Conselho Nacional de Desenvolvimento Científico e Tecnológico

EI – Extração da Informação

GRU – *Gated Rated Unit*

HTML – *Hypertext Markup Language*

IBICT – Instituto Brasileiro de Informação em Ciência e Tecnologia

JSON – *JavaScript Object Notation*

LSTM – *Long-Short Term Model*

PD&I – Pesquisa, Desenvolvimento e Inovação

PDF – *Portable Document Format*

Petrobras – Petróleo Brasileiro S.A.

PLN – Processamento de Linguagem Natural

PUC-Rio – Pontifícia Universidade Católica do Rio de Janeiro

RCNN – *Recurrent Convolutional Neural Network*

RNN – *Recurrent Neural Network*

TF-IDF – *Term Frequency – Inverse Document Frequency*

UFBA — Universidade Federal da Bahia

UFES — Universidade Federal do Espírito Santo

UFF — Universidade Federal Fluminense

UFMG — Universidade Federal de Minas Gerais

UFPE — Universidade Federal de Pernambuco

UFRGS — Universidade Federal do Rio Grande do Sul

UFRJ – Universidade Federal do Rio de Janeiro

UFRN — Universidade Federal do Rio Grande do Norte

UFS — Universidade Federal de Sergipe

UFSC — Universidade Federal de Santa Catarina

UFSCar -- Universidade Federal de São Carlos

UFV — Universidade Federal de Viçosa

UnB — Universidade de Brasília

UNESCO – Organização das Nações Unidas para a Educação, a Ciência e a Cultura

UNESP -- Universidade Estadual Paulista

UNICAMP -- Universidade Estadual de Campinas

URL – *Uniform Resource Locator*

USP — Universidade de São Paulo

XML – *Extensible Markup Language*

*Como todo possuidor de uma biblioteca,  
Aureliano se sabia culpado de não conhecê-la  
até o fim; essa controvérsia permitiu-lhe  
chegar a um acordo com muitos livros que  
pareciam censurar sua incúria.*

**Jorge Luis Borges, *Os Teólogos*.**

# 1

## Introdução

Algoritmos baseados em aprendizado de máquina são, por definição, dependentes de um conjunto de dados para a realização do seu treinamento. Não é diferente com os algoritmos de processamento de linguagem natural (PLN), que necessitam de uma grande quantidade de textos para conseguir extrair as relações linguísticas e semânticas dos dados apresentados. No entanto, há uma grande dificuldade para o desenvolvimento de modelos de PLN pela indústria do petróleo no Brasil. Há uma carência de bases de documentos de referência que possam servir como conjunto de dados para a realização desses treinamentos.

Portanto, o objetivo desse trabalho foi criar o Petrolês. O Petrolês é um corpus público formado por teses e dissertações em português, no domínio de óleo e gás. Esse corpus servirá como de referência para os grupos de pesquisas em inteligência artificial e empresas relacionadas à indústria do petróleo. Ele servirá para a extração de termos técnicos e jargões específicos do setor, bem como para o treinamento e teste de algoritmos de PLN especializados para a indústria do petróleo.

Para a criação desse corpus, foram utilizados documentos disponíveis na Biblioteca Digital de Teses e Dissertações (BDTD) do Instituto Brasileiro de Informação em Ciência e Tecnologia (IBICT). No entanto, como a BDTD possui quase 600 mil documentos de todas as áreas do conhecimento, foi necessário identificar se um documento era relevante para a indústria de óleo e gás antes de extraí-lo da base. Todo esse processo foi feito automaticamente utilizando um modelo treinado para classificar os resumos dos documentos e outro robô para processar centenas de milhares de teses e dissertações presentes na BDTD.

O Petrolês é composto por 4.302 documentos, contendo cerca de 6 milhões de sentenças e 74,4 milhões de tokens. Esse novo corpus servirá como referência para o desenvolvimento dos futuros trabalhos de processamento de linguagem na indústria do petróleo.

## 2

### Corpus

Todos os algoritmos de aprendizado de máquina dependem de dados para o seu treinamento. No caso dos algoritmos de processamento de linguagem, os dados de treinamento são conjuntos de textos. Um conjunto de textos, de forma geral, é chamado de corpus, e seu plural de corpora. Dependendo do objetivo, os corpora podem ter diversas características, por exemplo eles podem ser monolíngues ou multilíngues, podem ser transcrições da linguagem falada no dia a dia ou serem compostos de textos técnicos de uma área do conhecimento específica. A escolha ou criação de um corpus depende do uso e tarefas que se pretende.

Segundo *The Oxford Companion to the English*, corpus é:

Um corpo de textos, enunciados ou outros espécimes considerados mais ou menos representativos da linguagem, e usualmente armazenados em uma base de dados eletrônica. Atualmente, corpora computacionais podem armazenar muitos milhões de palavras corridas, no qual as suas características podem ser analisadas por meio de marcações (adição de marcas de identificação e classificação nas palavras e outras formações) e o uso de programas de concordância. [15] <sup>1</sup>

Os corpora podem ser classificados de acordo com o seu conteúdo, seus metadados, presença de multimídia e sua relação com outros corpora. De acordo com a Lexical Computing [9], podemos classificar os corpora da seguinte maneira:

- Corpus monolíngue – são aqueles compostos por textos em apenas um idioma. São os tipos de corpora mais comuns.
- Corpus paralelo – é composto por dois corpora monolíngue em idiomas diferentes, sendo que um é a tradução do outro. Os textos devem estar alinhados permitindo observar como as correspondências dos segmentos, em geral palavras e sentenças, aparecem nos dois idiomas.

<sup>1</sup>A body of texts, utterances, or other specimens considered more or less representative of a language, and usually stored as an electronic database. Currently, computer corpora may store many millions of running words, whose features can be analyzed by means of tagging (the addition of identifying and classifying tags to words and other formations) and the use of concordancing programs

- Corpus multilíngue – similar ao corpus paralelo, podendo ser composto por corpora em várias línguas. Um corpus multilíngue com apenas dois idiomas é idêntico a um corpus paralelo.
- Corpus comparável – é composto por dois ou mais corpus monolíngue cujos textos são relacionados ao mesmo tópico, no entanto não são traduções um do outro. Um exemplo são corpora composto por verbetes da Wikipédia.
- Corpus de aprendiz – é formado por textos produzidos por pessoas que estão aprendendo uma língua estrangeira. É utilizado para identificar os principais erros e dificuldades no aprendizado de uma nova língua.
- Corpus diacrônicos – contém documentos de diferentes períodos de tempo. É usado para o estudo do desenvolvimento ou mudança da linguagem.

Complementando essas categorias, um corpus também pode ser classificado como:

- Especializado – quando é formado por documentos exclusivos de uma área do conhecimento ou de um domínio específico.
- Multimídia – corpora formados por textos e enriquecidos com áudios, imagens e outros conteúdos multimídia.

Para as atividades de processamento de linguagem natural mais comuns, há algumas características desejáveis em um corpus. Uma delas é a profundidade, ou seja, é importante que a quantidade de palavras total do corpus bem como o tamanho do vocabulário (número de palavras únicas) sejam relativamente grandes. Um corpus muito “raso” pode não conseguir captar a diversidade linguística presente nos textos. Outra característica importante é a data de criação dos textos. Exceto para corpora diacrônicos, em geral, é desejável textos recentes. Os metadados que acompanham os textos podem trazer informações valiosas e podem auxiliar em diversas tarefas de aprendizado supervisionado. Para muitas aplicações também é necessária uma pluralidade de gêneros e tipos de texto, evitando que haja um viés proveniente de documentos de um mesmo tipo ou fonte. Finalmente, para algumas aplicações é útil que o corpus já esteja pré-processado e limpo. [19]

Alguns dos elementos básicos para o processamento automático da linguagem podem ser obtidos através de corpora bem curados e montados. Exemplos desses elementos são listas de palavras, classes gramaticais e modelos mais complexos de representação de palavras usando espaços vetoriais.

As palavras são os blocos básicos que constituem um corpus. Obter uma lista do vocabulário presente em um corpus, bem como a frequência em que as palavras aparecem pode ser útil em vários contextos. Por exemplo, conhecer quais são as *stopwords*, ou seja, palavras que trazem pouca informação e acrescentam pouco na predição ou recuperação da informação [13], é útil para diversas tarefas de limpeza e pré-processamento de textos. No entanto, além de conhecer as palavras, também pode ser interessante conhecer a estrutura gramatical das sentenças e textos. Por isso, muitos corpora são manualmente anotados com as estruturas gramaticais (*Part-of-speech tagging*) e com as relações entre essas estruturas (*Treebanks*). [19]

Finalmente, pode-se utilizar um corpus para se criar uma representação matemática das palavras de forma a capturar algumas relações semânticas presentes nos textos. Essas representações matemáticas, em geral vetores, podem servir de entrada para diversos modelos de aprendizado de máquina. Em 2013, Mikolov et al. [16] apresentaram o modelo de vetorização de palavras utilizando redes neurais mais utilizado atualmente, o Word2Vec. Nesse modelo, todas as palavras são representadas como vetores e os valores desses vetores é baseado nas palavras vizinhas. Ou seja, as palavras deixam de ser apenas representações gráficas e passam a capturar parte do contexto das sentenças onde estão inseridas. Desta forma, o conjunto de textos utilizados para treinar esses vetores é fundamental para a qualidade final dessa representação.

Mais recentemente, arquiteturas de *deep learning* mais complexas estão sendo testadas para capturar as relações de dependências entre as palavras em documentos escritos em linguagem natural. Atualmente, o modelo BERT (*Bidirectional Encoder Representations from Transformers*) [10] tenta capturar as relações semânticas do texto onde ele é treinado e, atualmente, tem alcançado os melhores resultados em diversas tarefas de processamento de linguagem natural. Assim como os modelos de vetorização de palavras, os modelos de *deep learning* são altamente dependentes da qualidade do corpus onde eles estão sendo treinados.

Além de se extrair relações linguísticas, muitos corpora também são montados para possibilitar o treinamento de modelos de aprendizado de máquinas em tarefas que se deseja automatizar. Algumas tarefas estão relacionadas a anotações automáticas das palavras de um texto. Pode-se treinar um modelo para identificar as classes gramaticais das palavras e as relações de dependências sintáticas nas sentenças, assim como identificar entidades relevantes, como nomes de pessoas, empresas ou locais por exemplo. Essas marcações automáticas podem ser muito úteis para desambiguar um texto ou gerar metadados de forma automática. Marcações automáticas, em geral, são etapas preliminares



do treinamento de tarefas mais complexas como classificação automática de textos, análise de sentimento ou até tarefas de perguntas e respostas.

A criação de corpora é uma atividade bem antiga e tem sido a base de muitos estudos linguísticos. Nas últimas décadas, com o crescimento dos recursos computacionais, cresceu muito a disponibilidade e o uso de corpora. Na tabela 2.1 é listado alguns dos corpora mais utilizados.

Tabela 2.1: Corpora mais comuns[2]

Nome	Descrição	Link
Reuters News Dataset	Notícias da agência Reuters do ano de 1987.	<a href="https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection">https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection</a>
IMDB Reviews	Avaliações de 2.500 filmes para análise de sentimento binária	<a href="http://ai.stanford.edu/~amaas/data/sentiment/">http://ai.stanford.edu/~amaas/data/sentiment/</a>
The WikiQA Corpus	Corpus com pares de perguntas e respostas em domínio geral.	<a href="https://www.microsoft.com/en-us/download/details.aspx?id=52419&amp;from=http%3A%2F%2Fresearch.microsoft.com%2Fapps%2Fmobile%2Fdownload.aspx%3Fp%3D4495da01-db8c-4041-a7f6-7984a4f6a905">https://www.microsoft.com/en-us/download/details.aspx?id=52419&amp;from=http%3A%2F%2Fresearch.microsoft.com%2Fapps%2Fmobile%2Fdownload.aspx%3Fp%3D4495da01-db8c-4041-a7f6-7984a4f6a905</a>
Sentiment140	Base com 160,000 tweets formatados com 6 atributos: polarização, ID, data do tweet, consulta, usuário e texto. Emoticons foram removidos.	<a href="http://help.sentiment140.com/for-students/">http://help.sentiment140.com/for-students/</a>
Twitter US Airline Sentiment	Tweets sobre companhias aéreas americanas, recuperados em fevereiro de 2015, classificados como positivo, negativo ou neutro.	<a href="https://www.kaggle.com/crowdflower/twitter-airline-sentiment">https://www.kaggle.com/crowdflower/twitter-airline-sentiment</a>
Spoken Corpora Wikipédia	Centenas de horas de áudio contendo artigos da Wikipédia lidos em inglês, alemão e holandês.	<a href="https://nats.gitlab.io/swc/">https://nats.gitlab.io/swc/</a>
Wikipédia Links Data	13 milhões de documentos composto por páginas web contendo pelo menos um hiperlink apontando para a Wikipédia em inglês. Cada página da Wikipédia é considerada uma entidade e os links representam uma menção as entidades.	<a href="https://code.google.com/archive/p/wiki-links/downloads">https://code.google.com/archive/p/wiki-links/downloads</a>
Gutenberg List eBooks	Lista anotada de ebooks do Projeto Gutenberg contendo as informações básicas dos livros e organizado por ano.	<a href="http://www.gutenberg.org/wiki/Gutenberg:Offline_Catalogs">http://www.gutenberg.org/wiki/Gutenberg:Offline_Catalogs</a>
Jeopardy	O arquivo contém mais de 200.000 perguntas e respostas do programa Jeopardy. Cada ponto de dado possui outras informações complementares como categoria da pergunta, número do programa e data.	<a href="https://www.reddit.com/r/datasets/comments/1uyd0t/200000_jeopardy_questions_in_a_json_file/">https://www.reddit.com/r/datasets/comments/1uyd0t/200000_jeopardy_questions_in_a_json_file/</a>
European Parliament Proceedings Parallel Corpus	Pares de sentenças de proveniente de processos do Parlamento Europeu em 21 idiomas europeus.	<a href="http://statmt.org/europarl/">http://statmt.org/europarl/</a>

Em português, a disponibilidade de corpora para treinamento de modelos para processamento de linguagem natural é muito menor do que em inglês ou outras línguas. A tabela 2.2 mostra alguns exemplos dos corpora mais usados em português.

Tabela 2.2: Corpora mais comuns em português [17]

Nome	Descrição	Link
O Corpus do Português	Corpus contendo um bilhão de palavras em português.	<a href="https://www.corpusdoportugues.org/">https://www.corpusdoportugues.org/</a>
NILC / São Carlos Corpora	Vários corpora com texto em português contemporâneo, todos com POS-Tagging.	<a href="https://www.linguatca.pt/acesso/corpus.php?corpus=SAO CARLOS">https://www.linguatca.pt/acesso/corpus.php?corpus=SAO CARLOS</a>
Tweets em português para análise de sentimento	800 mil tweets em português, divididos em positivo, negativo e neutro.	<a href="https://www.kaggle.com/augustop/portuguese-tweets-for-sentiment-analysis">https://www.kaggle.com/augustop/portuguese-tweets-for-sentiment-analysis</a>
BlogSet-BR	Um extensivo corpus com 2.1 bilhões de palavras provenientes de 808 blogs e 7,4 milhões de posts escritos em português brasileiro.	<a href="http://www.inf.pucrs.br/linatural/wordpress/recursos-e-ferramentas/blogset-br/">http://www.inf.pucrs.br/linatural/wordpress/recursos-e-ferramentas/blogset-br/</a>

Finalmente, para muitas aplicações é mais interessante trabalhar com corpora de domínio específicos do que de domínio geral. É possível extrair

termos técnicos e jargões de documentos de uma área específica que dificilmente apareceriam em textos de amplo domínio. Da mesma maneira, algumas palavras possuem significados ou são usadas diferentemente em áreas do conhecimento distintas. Dois exemplos de corpora da área da biologia são o *BioCreative*<sup>2</sup> e o *GENIA*<sup>3</sup>.

Neste trabalho, o objetivo foi criar o Petrolês, um corpus monolíngue especializado no domínio de óleo e gás. Uma rica fonte de textos técnicos, de diversas áreas do conhecimento e públicos são os repositórios e bibliotecas de instituições de pesquisas e universidades. Por esse motivo, teses e dissertações foram os documentos que serviram de fonte de textos para o corpus criado.

<sup>2</sup><https://biocreative.bioinformatics.udel.edu/about/background/description/>

<sup>3</sup>[https://academic.oup.com/bioinformatics/article/19/suppl\\_1/i180/227927](https://academic.oup.com/bioinformatics/article/19/suppl_1/i180/227927)

### 3

## Teses e Dissertações em Óleo e Gás

Para preencher a lacuna que existe na disponibilidade de um corpus público em português no domínio da indústria do petróleo, o Petrolês foi criado com base em teses e dissertações acadêmicas. Para esse fim, utilizamos como fonte a Base Digital de Teses e Dissertações (BDTD)<sup>1</sup> do Instituto Brasileiro de Informação de Ciência e Tecnologia (IBICT). Nessa base estão disponíveis quase 600.000 documentos das principais instituições de pesquisa do Brasil.

A escolha das teses e dissertações para serem os componentes básicos do Petrolês se deu pelos seguintes motivos. Primeiramente, teses e dissertações são resultados de pesquisas de mestrado e doutorados e são públicas. Essa característica facilita o compartilhamento desse corpus entre diversos grupos de pesquisa e torná-lo uma referência entre os corpora em português. Por serem documentos técnicos, esse tipo de documento possui uma variedade de termos e jargões específicos das áreas de conhecimento a que pertencem. Uma outra vantagem é o fato de serem documentos relativamente bem padronizados, as diagramações das páginas são relativamente simples o que facilita a extração dos textos dos documentos. Por fim, teses e dissertações são documentos longos quando comparados à documentos presentes em outros corpora, como por exemplo *tweets*, verbetes da Wikipédia e artigos jornalísticos.

A principal desvantagem da utilização das teses e dissertações está no fato desses documentos estarem no formato PDF (*Portable Document Format*). Buscar e extrair informações rápida e eficientemente de documentos PDF ainda é um problema a ser resolvido, esta é uma das principais áreas de pesquisa do ramo conhecido como Extração da Informação (EI) [22]. Outra desvantagem de um corpus formado por documentos acadêmicos está no fato dele conter jargões e estilos de escrita particulares da academia. Pode haver alguma incompatibilidade entre a linguagem utilizada nas teses e dissertações quando comparado aos documentos utilizados no dia a dia da indústria.

A Base Digital de Teses e Dissertações (BDTD) é referência no acesso livre ao conhecimento e reconhecida por ser uma das maiores bibliotecas do mundo em número de teses e dissertações de um só país. A origem do Instituto Brasileiro de Informação de Ciência e Tecnologia remonta à década de 50, quando a UNESCO incentivou a criação no Brasil de um centro nacional de bibliografia. A BDTD foi criada em 2002 e hoje integra o sistema de informação de teses e dissertações das instituições de ensino e pesquisa brasileiras. [12]

<sup>1</sup><http://bdttd.ibict.br/vufind/>

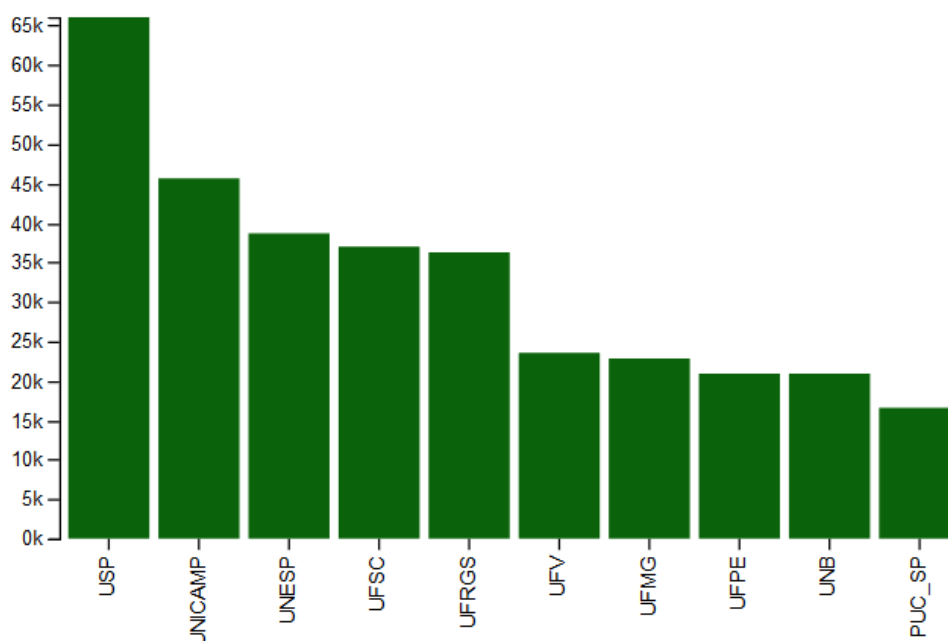


Figura 3.1: Quantidade de documento por instituição [11]

Em 2019, a BDTD contava com 586.225 documentos cadastrados em sua base, provenientes de 116 instituições diferentes. Desses documentos, 431.225 eram dissertações de mestrado e 155.002 eram teses de doutorado. Anualmente, são depositados cerca de 40 mil novos documentos, em sua grande maioria em português. As universidades públicas, tanto federais quanto estaduais, são as grandes depositárias da BDTD. (figura 3.1)

O Petrolês, portanto, foi composto por documentos cadastrados na BDTD. Primeiramente, foi necessário identificar, dentre todas as teses e dissertações existentes, quais eram as mais relevantes para montar um corpus em português de óleo e gás. No entanto, a BDTD só possui os resumos das teses e dissertações, os documentos completos estão nos repositórios de cada uma das instituições. Portanto, seria muito difícil buscar de forma automática em todas as 116 instituições presentes da BDTD.

A estratégia utilizada foi escolher as instituições que teriam a maior chance de ter produzido teses e dissertações na área de conhecimento alvo. O primeiro critério utilizado foi escolher as 9 instituições com mais documentos depositados na BDTD. Os destaques, segundo esse critério, foram USP e UNICAMP com mais de 45 mil documentos cada uma, e UNESP, UFSC e UFRGS com quase 40 mil cada. [11]

O segundo critério utilizado foi escolher instituições com Unidades de Pesquisas na área de óleo e gás. Para tal, foi utilizado como referência o credenciamento feito pela Agência Nacional de Petróleo, Gás Natural e Biocombustível (ANP) às instituições de pesquisa e desenvolvimento. Esse

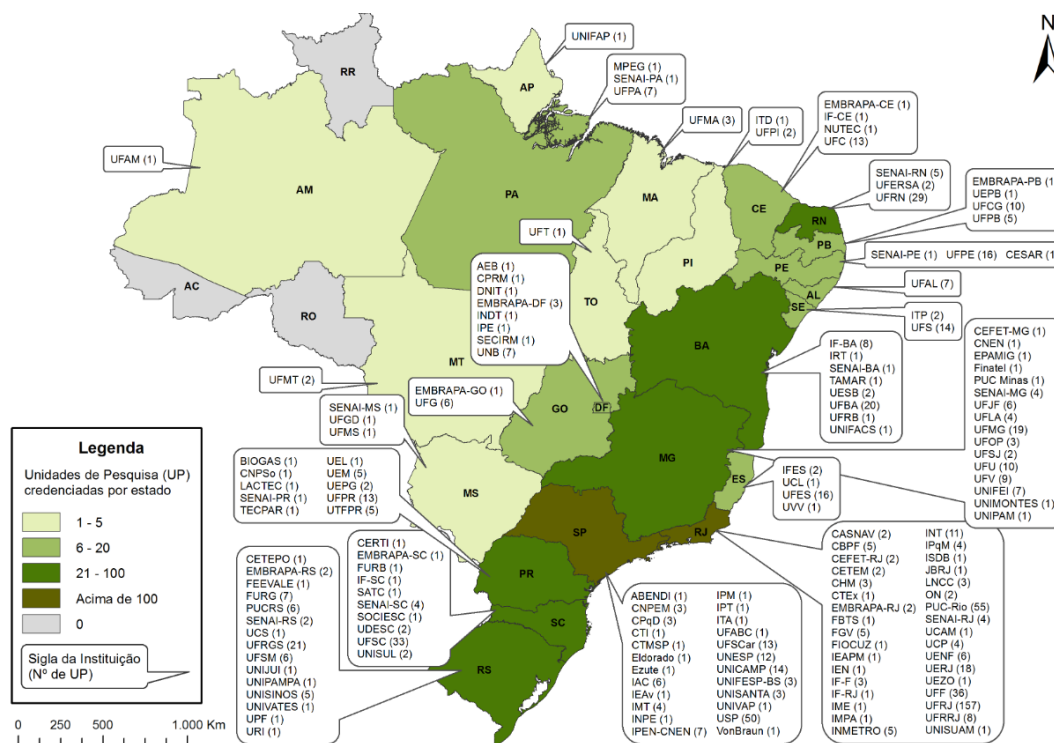


Figura 3.2: Instituições Credenciadas pela ANP até 30 de setembro 2019 [3]

credenciamento é o reconhecimento formal da agência reguladora de que a unidade de pesquisa possui condições técnicas e infraestrutura necessária para a execução de projetos com recursos da cláusula de PD&I<sup>2</sup>. Até setembro de 2019, foram credenciadas 150 instituições que, em conjunto, possuíam 898 unidades de pesquisas nas áreas de interesse da ANP (Figura 3.2). Escolhemos as instituições que possuíam mais de 10 unidades de pesquisas credenciadas.

[3]

Por fim, algumas instituições que se enquadravam nos critérios levantados não puderam ser incluídas devido a dificuldades técnicas em recuperar os documentos dos seus repositórios institucionais. Dois casos notáveis foram a UFRJ e PUC-Rio. Essas duas instituições, por exemplo, possuíam repositórios em plataformas que dificultavam a recuperação automática ou exigiam *login* e senha para acesso. Em suma, as instituições consideradas foram:

- UFBA – Universidade Federal da Bahia
- UFES – Universidade Federal do Espírito Santo
- UFF – Universidade Federal Fluminense
- UFMG – Universidade Federal de Minas Gerais

<sup>2</sup>A regulamentação da aplicação dos recursos da cláusula de PD&I podem ser encontrados na Resolução ANP nº 47/2012 e no Regulamento Técnico ANP nº 7/2012. Em: <http://www.anp.gov.br/pesquisa-desenvolvimento-e-inovacao/investimentos-em-p-d-i/regulamentacao-tecnica-relativa-aos-investimentos-em-p-d-i>

- UFPE – Universidade Federal de Pernambuco
- UFRGS – Universidade Federal do Rio Grande do Sul
- UFRN – Universidade Federal do Rio Grande do Norte
- UFS – Universidade Federal de Sergipe
- UFSC – Universidade Federal de Santa Catarina
- UFSCar – Universidade Federal de São Carlos
- UFV – Universidade Federal de Viçosa
- UnB – Universidade de Brasília
- UNESP – Universidade Estadual Paulista
- UNICAMP – Universidade Estadual de Campinas
- USP – Universidade de São Paulo

O objetivo final do Petrolês é a utilização dos textos das teses e dissertações para tarefas de processamento de linguagem natural. No entanto esses documentos possuem outras informações além de textos puros. Em geral, uma tese ou dissertação é composta por algumas páginas iniciais contendo capa, agradecimentos e sumários, da mesma forma as últimas páginas são compostas por referências bibliográficas e anexos (Figura 3.3). Apesar de conter texto, o conteúdo dessas páginas costuma ser ruim para a montagem do corpus por possuírem pouco texto corrido ou textos com pouca relevância.

Os capítulos dos documentos são o alvo da montagem do corpus. Neles são encontrados os principais termos técnicos e a maior quantidade de texto. Entretanto, os capítulos não são formados exclusivamente por textos, neles também são encontradas muitas figuras, tabelas e fórmulas (Figura 3.4). Apesar das figuras, tabelas e fórmulas serem importante para a leitura, para a montagem de um corpus esses elementos podem atrapalhar diversos algoritmos de aprendizado de máquina. A extração das tabelas dos documentos, por

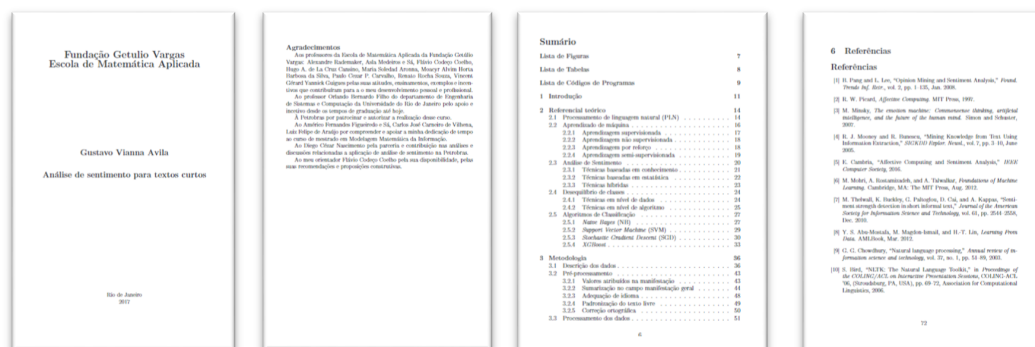


Figura 3.3: Páginas iniciais e finais: Capa, agradecimentos, sumários e bibliografia [4]

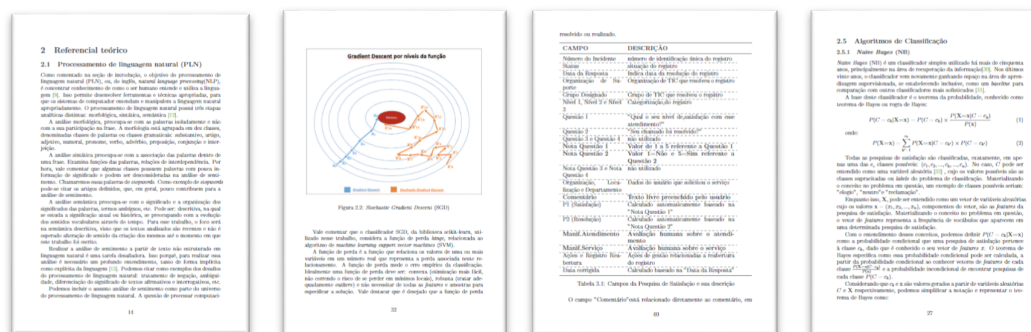


Figura 3.4: Conteúdo dos capítulos: Texto, figuras, tabelas e fórmulas [4]

exemplo, pode resultar em um texto ruidoso, onde o texto do capítulo acaba misturado com as linhas das tabelas. Neste caso, é mais interessante eliminar a tabelas e utilizar somente o texto para o treinamento. O mesmo pode ocorrer com as fórmulas e figuras.

Por fim, além do conteúdo do documento propriamente dito, também faz parte do corpus informações referentes aos documentos, ou seja, os metadados. Para as teses e dissertações extraídas da BDTD, é possível recuperar os seguintes metadados:

- Título
- Nível de acesso, em geral aberto
- Data de defesa
- Autor/a
- Orientador/a
- Coorientador/a
- Tipo de documento, tese ou dissertação
- Idioma, maioria em português
- Instituição de defesa
- Programa
- Assuntos em português
- Áreas de conhecimento
- Link para download do texto completo
- Resumo em português:
- Resumo em inglês
- Banca

A escolha da BDTD proporcionou uma farta fonte de documentos, no entanto, também acrescentou o desafio de escolher aqueles mais relevantes. Por esse motivo, a classificação das teses e dissertações foi uma das etapas mais importantes da criação do Petrolês.



## 4

### Classificação de Textos

A Base Digital de Teses e Dissertações (BDTD) do IBICT é um repositório que contém documentos de todas as áreas do conhecimento. Portanto, foi necessário selecionar quais documentos eram relevantes para entrar no Petrolês e quais deveriam ser deixados de fora. A maneira mais direta de realizar essa tarefa seria solicitar que especialistas lessem os resumos dos documentos e decidissem se eles eram relevantes ou não para a indústria do petróleo. No entanto essa tarefa seria extremamente exaustiva e demorada. A abordagem utilizada, portanto, foi a utilização de algoritmos de classificação de textos para realizar essa tarefa.

A classificação de textos, também conhecida por categorização ou identificação de tópicos, é a tarefa de classificar um texto aleatório no seu respectivo domínio. Essa classificação pode ser binária, ou seja, identificar se um dado texto pertence ou não a determinado domínio, ou multi classe, quando há diversas classes possíveis. Os problemas de classificação também podem ser divididos em *single* ou *multi-label*, quando são atribuídas, respectivamente, um ou várias classes para cada texto. [18]

Para a realização das tarefas de *text mining* e de processamento de linguagem natural, primeiramente, é necessário extrair os atributos que serão utilizados pelos algoritmos de aprendizado de máquina. Em geral, as palavras são utilizadas como unidades básicas a serem tratadas como atributos dos documentos. As palavras podem ser tratadas isoladamente, uni-gramas, ou em conjunto, bi-gramas, tri-gramas ou n-gramas. Outros elementos do texto também poderiam ser considerados atributos, como letras, números, caracteres especiais, sentenças e parágrafos.

Uma forma clássica de representar um texto é conhecida por *bag-of-words*. Neste tipo de representação, cada documento é representado como um vetor. O tamanho desse vetor é igual ao tamanho do vocabulário e os valores representam a frequência que cada palavra aparece no documento [8]. Outra representação bastante utilizada é a TF-IDF (*Term Frequency – Inverse Document Frequency*). Nele o valor de cada palavra, ou n-gram, é dado pela sua exaustividade, o quão frequente um termo se apresenta em um documento (TF), e especificidade, o quão raro esse termo aparece na coleção de documentos (IDF) [5]. Assim como na representação *bag-of-words*, cada documento pode ser representado por um vetor do tamanho do vocabulário, cujos valores são os índices TF-IDF de cada termo para cada documento.

Uma das dificuldades das representações *bag-of-words* e TF-IDF é a necessidade de se trabalhar com longos vetores esparsos, ou seja, os vetores podem ter dezenas de milhares de valores representados, mas quase todos são zeros. Outra característica é a ausência de relação entre os termos, nem a ordem em que as palavras aparecem no texto nem a similaridades entre as palavras são capturadas por essas representações. Esses são justamente os problemas atacados pela representação de palavras por vetores obtidas a partir de treinamento de redes neurais, também conhecida por Word2Vec [16]. No Word2Vec, bem como em outras formas de word embeddings, cada palavra é representada por um vetor de tamanho predefinido e treinados previamente.

Para a classificação dos textos foram testados diversos algoritmos de aprendizado de máquina. Primeiramente, foram utilizados algoritmos clássicos como *Naive Bayes*, Regressão Logística, *Support Vector Machine*, *Bagging* e *Boosting Models*. Em seguida, também foram testados os algoritmos baseados em redes neurais, tanto redes rasas quanto redes profundas, como as redes convolucionais, *Long-Short Term Model* (LSTM), *Gated Rated Unit* (GRU), bidirecionais e redes convolucionais recorrente.

Dentre os algoritmos clássicos, o primeiro algoritmo testado foi o *Naive Bayes*. É o algoritmo baseado no teorema de Bayes e é particularmente recomendado para alta dimensionalidade. Em seguida, foi a vez da Regressão Logística, método onde as probabilidades dos resultados são descritas como uma função das variáveis observadas. Já o *Support Vector Machine* é um algoritmo que encontra um hiperplano que separa linearmente os dados de treinamento de acordo com as classes apresentadas. Finalmente, os modelos de *Bagging* e *Boosting* agrupam diversos classificadores mais simples para gerar um classificador mais preciso. [1]

Já para os classificadores baseados em redes neurais, foram utilizadas tanto as redes com poucas camadas, as chamadas *multi layer perceptron*, como os algoritmos de *deep learning*. Dos modelos de *deep learning* o primeiro a ser testado foi a rede convolucional, que faz uso de camadas convolucionais para extrair diferentes atributos dos dados. Em seguida, foram testados alguns modelos de redes neurais recorrentes (RNN), ou seja, modelos que levam em consideração a sequência de entrada dos dados, algo muito importante em textos. Foram testadas duas arquiteturas de redes recorrentes a *Long-Short Term Model* (LSTM) e a *Gated Rated Unit* (GRU). Também foi testada uma rede bidirecional, arquitetura de rede recorrente que usa os dois sentidos possíveis de entrada do texto. Por fim, também foi testado uma arquitetura tanto com camadas convolucionais quanto com camadas recorrentes (RCNN). [21]

O classificador de texto foi o bloco fundamental para a criação do Petrolês. Ele permitiu identificar facilmente, dentre centenas de milhares de possibilidades, quais os documentos eram relevantes para compor o corpus de óleo e gás. Uma vez que a fonte de documentos e os algoritmos de classificação estavam disponíveis, foi possível realizar os experimentos para treinar o melhor classificador.

## 5 Metodologia

Para a criação do Petrolês foram necessárias diversas etapas. Inicialmente, foi preciso ler os resumos das teses e dissertações presentes na BDTD. Portanto, foi criado um *crawler*<sup>1</sup> capaz de acessar automaticamente esses documentos. Em seguida, um algoritmo classificador de texto foi treinado para reconhecer se um determinado resumo pertencia a um documento relevante ao setor de petróleo. Com esse classificador em mãos, foi possível utilizar o *crawler* para ler todos os documentos das instituições selecionadas e identificar se eles eram relevantes ou não para a montagem do corpus. Uma vez identificados os documentos relevantes, foi criado um outro conjunto de *crawlers*, um para cada instituição de pesquisa, com o objetivo de coletar os arquivos originais das teses e dissertações nos seus respectivos repositórios institucionais. Finalmente, após todos os documentos coletados, foi possível extrair apenas os textos dos documentos para se criar um corpus do domínio de óleo e gás em português. (Figura 5.1)

### 5.1 Crawler da BDTD

A primeira tarefa realizada foi a criação de um *script* que possibilitasse a coleta automática dos metadados de um documento presente na BDTD. Cada documento é apresentado em uma página com uma URL própria. A primeira

<sup>1</sup>Um programa que automaticamente acessa diversas páginas da internet coletando informações.

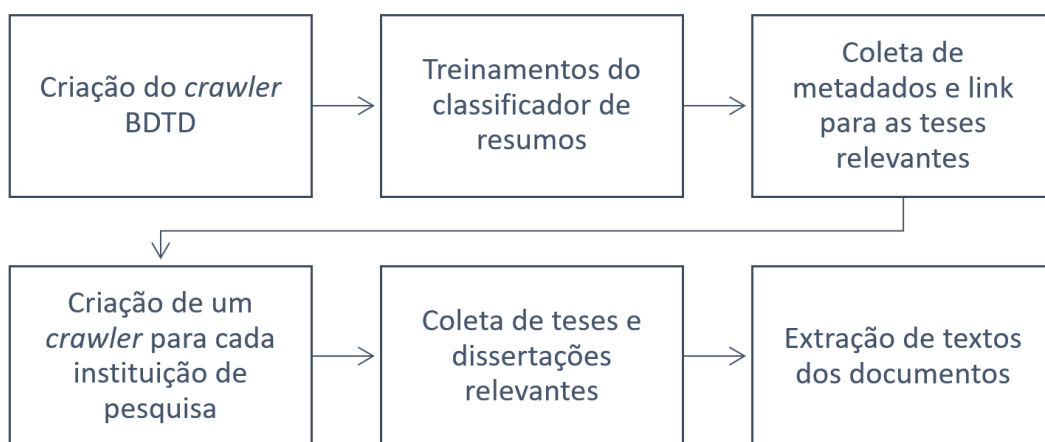


Figura 5.1: Etapas para a criação de um corpus de textos no domínio de óleo e gás em português. (Fonte: Elaborada pelo autor)

Nível de Acesso:	openAccess
Data de Defesa:	2018
Autoria:	Pinto, Guilherme Franco Silva
Orientadora:	Furnival, Ariadne Chloe Mary
Tipo Documento:	Dissertação
Idioma:	por
Instituição de Defesa:	Universidade Federal de São Carlos Câmpus São Carlos
Programa:	Programa de Pós-graduação em Ciência da Informação
Assuntos em Português:	Comportamento informacional Mineração textual Twitter Direito autoral
Assuntos em Inglês:	Information behavior Twitter Text mining Copyright
Áreas de Conhecimento:	CIENCIAS SOCIAIS APLICADAS > CIENCIA DA INFORMACAO
Download Texto Completo:	<a href="https://repositorio.ufscar.br/handle/ufscar/10535">https://repositorio.ufscar.br/handle/ufscar/10535</a>
<div> <div>Descrição</div> <div> <p><b>Resumo Português:</b></p> <p>Este trabalho busca explorar o comportamento informacional dos usuários de mídias sociais, especificamente o Twitter, quando buscam e compartilham informações sobre Direito Autoral e suas vertentes. Tendo em vista as mudanças sociais, culturais e econômicas que acarretaram o uso das mídias sociais, exploramos como as adaptações da legislação de Direito Autoral às novas plataformas digitais afetam a utilização destas plataformas por seus usuários e como os usuários buscam informações relacionadas. Utilizamos o API do Twitter para coletar as postagens (tweets) feitas na plataforma que falem sobre Direito Autoral e suas áreas afins, como licenças livres, Creative Commons, domínio público etc. A seguir, utilizando análises de Mineração Textual dentro do ambiente para computação estatística R, foi possível examinar e relacionar os termos e palavras das postagens com as dúvidas e perguntas mais comuns sobre Direito Autoral. Finalmente, foi possível identificar exemplos de comportamento informacional dos usuários do Twitter durante suas interações com outros usuários e as informações disponíveis no Twitter.</p> <p><b>Resumo inglês:</b></p> <p>This work aims to explore the information behavior of social media users, on Twitter specifically, at the moment they are searching and sharing information on Copyright Laws and its related subjects. Taking in consideration the social, cultural and economic changes that social media has made possible, this project will explore how the adapted Copyright Laws to digital platforms affect the usability of these platforms, and how its users seek information related to those adaptations. The Twitter API was used to collect the posts (tweets) made on the platform that are about Copyright and its related subjects, such as open licenses, Creative Commons, public domain, etc. Then, using Text Mining analysis in the statistical computing environment R, it was possible to assess and relate the terms and words used on these posts with the most common doubts and questions regarding Copyright. Finally, in this data it was possible to identify instances of information behavior of Twitter users during their interactions with each other and the information available on Twitter.</p> </div> </div>	

Figura 5.2: Exemplo de metadados no *website* da BDTD. [20]

parte da URL<sup>2</sup> sempre começa com `http://bdtb.ibict.br/vufind/Record`, já a última parte é composta por um identificador único do documento na BDTD. Cada página de uma tese ou dissertação possui uma tabela com metadados como o exemplo da Figura 5.2.

É possível recuperar os metadados das tabelas com o auxílio das suas respectivas marcações HTML<sup>3</sup>, como na Figura 5.3.

Para a criação do *script* de recuperação dos metadados da BDTD foram usadas, principalmente, duas bibliotecas em *Python*, *Request*<sup>4</sup> e *BeautifulSoup*<sup>5</sup>. Essas bibliotecas são respectivamente para acessar páginas de internet e

<sup>2</sup> *Uniform Resource Locator*

<sup>3</sup> *Hypertext Markup Language*

<sup>4</sup> <https://requests.readthedocs.io/en/master/>

<sup>5</sup> <https://www.crummy.com/software/BeautifulSoup/>

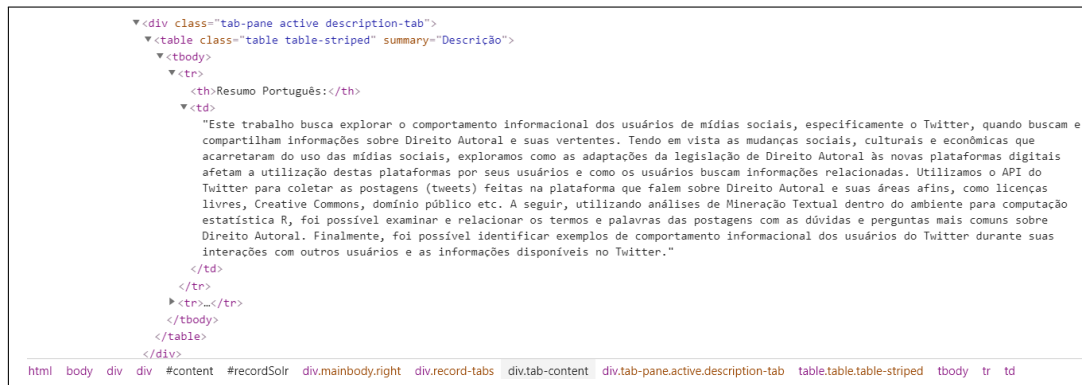


Figura 5.3: Exemplo do código HTML na BDTD. [20]

analisar documentos em formatos HTML, XML<sup>6</sup> e JSON<sup>7</sup>, dentre outros. Por fim, para recuperar todos os metadados de um documento, dada uma URL específica, foi criada a seguinte função:

---

**Código 1:** Função para coletar metadados de um documento na BDTD

---

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 #função para buscar os metadados das teses no BDTD
8 def tese_link(link):
9     #definir url
10    url = 'http://bdtd.ibict.br' + link
11
12    #Requisitar html e fazer o parser
13    f = requests.get(url, proxies = proxies).text
14    soup = bs(f, "html.parser")
15
16    #Dicionário para armazenar as informações da tese
17    tese = {}
18
19    #Adicionar título
20    tese['Title'] = soup.find('h3').get_text()
21    for doc in soup.find_all('tr'):
22        #Identificar atributo
23        try:
24            atributo = doc.find('th').get_text()
25        except:
26            pass
27        #Verificar se o atributo possui mais de um dado
28        for row in doc.find_all('td'):
29            #Adicionar o atributo no dicionário
```

<sup>6</sup>Extensible Markup Language

<sup>7</sup>JavaScript Object Notation

```
30         if row.find('div') == None:
31             try:
32                 tese[atributo] = doc.find('td').get_text()
33             except:
34                 pass
35         else:
36             element = []
37             #No dicionário, adicionar todos os dados ao seu
38             respectivo atributo
39             for e in doc.find_all('div'):
40                 try:
41                     sub_e = []
42                     for sub_element in e.find_all('a'):
43                         element.append(sub_element.get_text())
44                         #element.append(sub_e)
45                     except:
46                         pass
47                 tese[atributo] = element
48             return(tese)
```

---

Finalmente, esses metadados foram armazenados em um dicionário para posterior exportação em formato JSON.

## 5.2

### Classificador de textos

Para criar um classificador de documentos, ou seja, um algoritmo capaz de receber um texto aleatório e ser capaz de identificar se ele é relevante para o domínio de petróleo, primeiramente, foi preciso obter exemplos para treinamento. Foi necessário, portanto, criar uma base de treinamento com textos classificados como relevantes e não relevantes.

Para a criação dessa base de treinamento, foi utilizado uma estratégia para construir o conjunto de documentos relevantes e outra estratégia para os documentos não relevantes ao setor de petróleo. Foi utilizada uma abordagem bem direta para montar a base de documento relevantes, buscou-se na BDTD os documentos cujo assunto continha os termos “petroleo” e “gas natural”. Através dessa busca foi possível recuperar 1.986 documentos. Uma abordagem complementar foi a identificação de funcionários da Petrobras<sup>8</sup> que haviam depositado alguma tese ou dissertação no seu repositório institucional. Através dessa lista, buscou-se na BDTD os documentos desses autores. Com isso foi possível acrescentar mais 504 documentos ao conjunto de teses e dissertações do domínio de óleo e gás.

<sup>8</sup>Petróleo Brasileiro S.A. Maior empresa brasileira do setor de petróleo.

Para montar a base de teses e dissertações que não fazem parte do domínio de óleo e gás foi utilizada as áreas do conhecimento do CNPQ<sup>9</sup>, um dos metadados presentes na BDTD. Para manter a abrangência em diversos tipos de textos, foi escolhido 200 teses de cada área do conhecimento. Desta forma, foi montada uma base de treinamento com 2.490 documentos dentro do domínio de óleo e gás e 11.532 documentos de outras áreas do conhecimento.

O pré-processamento e a classificação foram baseados no trabalho de Bansal[6]. Primeiramente, todos os textos foram limpos e pré-processados. Em seguida, foram gerados os atributos de entradas necessários para os algoritmos de classificação. Com os atributos disponíveis, diversos classificadores foram treinados e avaliados.

O pré-processamento e a engenharia de atributos são as etapas na qual o texto é preparado para ser usado pelos algoritmos de classificação. Inicialmente, foi verificado se os documentos extraídos, usando as diversas estratégias de coleta na BDTD, não se repetiam. A etapa seguinte foi a separação entre os textos em português e em outro idioma, em geral inglês. Apesar da grande maioria dos documentos da BDTD estarem em português, há uma pequena parcela em inglês. Um fato bastante comum encontrado nos documentos foi a concatenação dos textos dos campos “Resumo Português” e “Resumo Inglês”. A mistura dos dois idiomas “sujava” os textos, piorando a acurácia do classificador. Foi necessário, portanto, a utilização da biblioteca *Langdetect*<sup>10</sup> para separar os resumos. Os textos sem resumos em português foram excluídos.

Na etapa seguinte, foram feitas as alterações no texto propriamente dito. Todas as letras foram deixadas em minúscula, as acentuações e as *stopwords* foram extraídas. A base de treinamento ficou com um grande desbalanceamento entre as duas classes. Portanto, foram sorteados para treinamento uma quantidade de documentos que deixasse o conjunto de treinamento balanceado. Desta forma, foram usados 4.242 resumos, 2.121 textos de cada classe. Destes, 80% foram usados para o treinamento dos modelos de classificação e os 20% restantes serviram para a avaliação.

Após os textos serem limpos e tratados, foram utilizadas algumas técnicas de extração de atributos dos textos. A primeira, e de certa forma a mais simples, foi transformar os textos em um vetor de frequência. Também foi utilizada a transformação do texto para vetores TF-IDF. Foram gerados três tipos de vetores TF-IDF, um com a representação de cada palavra, outro com a representação dos bigramas e trigramas, e finalmente, uma última representação no nível dos caracteres. Todas as transformações foram feitas

<sup>9</sup>Conselho Nacional de Desenvolvimento Científico e Tecnológico

<sup>10</sup><https://pypi.org/project/langdetect/>



utilizando a biblioteca *Sklearn*<sup>11</sup>.

---

**Código 2:** Criando os vetores de frequência de palavras e os vetores TF-IDF

---

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 # Criando um objeto Count Vector
8 count_vect = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}'
9                               ')
10
11 # Transforma os dados de treino e teste usando o objeto Count
   Vector
12 xtrain_count = count_vect.transform(train_x)
13 xtest_count = count_vect.transform(test_x)
14
15
16 # In[ ]:
17
18
19 # word level tf-idf
20 tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}'
21                               ', max_features=5000)
22 tfidf_vect.fit(tese_train['Resumo Português:'])
23 xtrain_tfidf = tfidf_vect.transform(train_x)
24 xtest_tfidf = tfidf_vect.transform(test_x)
25
26 # ngram level tf-idf
27 tfidf_vect_ngram = TfidfVectorizer(analyzer='word', token_pattern=r'
   \w{1,}', ngram_range=(2,3), max_features=5000)
28 tfidf_vect_ngram.fit(tese_train['Resumo Português:'])
29 xtrain_tfidf_ngram = tfidf_vect_ngram.transform(train_x)
30 xtest_tfidf_ngram = tfidf_vect_ngram.transform(test_x)
31
32 # characters level tf-idf
33 tfidf_vect_ngram_chars = TfidfVectorizer(analyzer='char',
34                                           token_pattern=r'\w{1,}', ngram_range=(2,3), max_features=5000)
35 tfidf_vect_ngram_chars.fit(tese_train['Resumo Português:'])
36 xtrain_tfidf_ngram_chars = tfidf_vect_ngram_chars.transform(train_x)
37 xtest_tfidf_ngram_chars = tfidf_vect_ngram_chars.transform(test_x)

```

---

A última transformação de texto para vetores foi feita utilizando o algoritmo *Word2Vec*. Primeiramente, foi treinado um modelo vetorial de palavras usando todos os textos disponíveis, ou seja, não foi necessário ter uma

<sup>11</sup>[https://scikit-learn.org/stable/modules/feature\\_extraction.html](https://scikit-learn.org/stable/modules/feature_extraction.html)

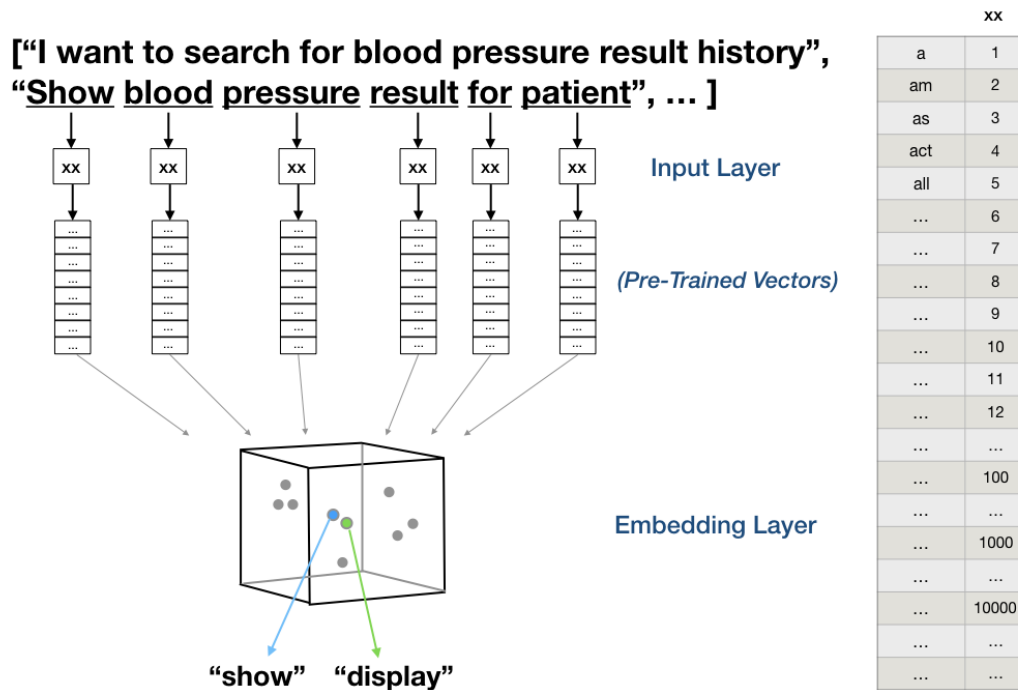


Figura 5.4: Representação de textos usando vetorização de palavras. [14]

base balanceada. Com isso obtivemos um modelo onde era possível mapear cada palavra do dicionário para um vetor correspondente (Figura 5.4). Esses vetores foram utilizados para representar os documentos.

---

### Código 3: Treinando e indexando os textos com o modelo Word2Vec

---

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 # Indexando as palavras presentes no modelo Word2Vec
8 word2index = {}
9 for index, word in enumerate(BDTD_word2vec_50.wv.index2word):
10     word2index[word] = index
11
12
13 # In[ ]:
14
15
16 # Indexando as palavras presentes no modelo Word2Vec
17 word2index = {}
18 for index, word in enumerate(BDTD_word2vec_50.wv.index2word):
19     word2index[word] = index
20
21 # Função para indexar o texto usando os índices do modelo Word2Vec
22 def index_pad_text(text, maxlen, word2index):

```

```

23     maxlen = 400
24     new_text = []
25     for sent in text:
26         temp_sent = []
27         for word in word_tokenize(sent):
28             try:
29                 temp_sent.append(word2index[word])
30             except:
31                 pass
32         # Estebelecendo um limite máximo de palavras para cada
           resumo (padding)
33         if len(temp_sent) > maxlen:
34             temp_sent = temp_sent[:400]
35         else:
36             temp_sent += [0] * (maxlen - len(temp_sent))
37         new_text.append(temp_sent)
38
39     return np.array(new_text)
40
41 maxlen = 400
42 train_seq_x = index_pad_text(train_x, maxlen, word2index)
43 test_seq_x = index_pad_text(test_x, maxlen, word2index)

```

Com os atributos criados, foi possível treinar os modelos de classificação usando diversas combinações de algoritmos e atributos. Para tal, foi criada uma função que permitisse entrar com os atributos e os tipos de classificadores. A função dispara o treinamento e calcula a acurácia usando a base de teste.

---

#### Código 4: Função para treinar modelo de classificação de textos

---

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[ ]:
5
6
7  def train_model(classifier, feature_vector_train, label,
           feature_vector_test, is_neural_net=False):
8      # fit the training dataset on the classifier
9
10     if is_neural_net:
11         callbacks = EarlyStopping(monitor='val_acc', patience=10,
                                   restore_best_weights=True)
12         history = classifier.fit(feature_vector_train,
13                                 label, #to_categorical(label),
14                                 epochs=1000,
15                                 batch_size=64,
16                                 validation_split=0.25,
17                                 callbacks=[callbacks])
18
19     # plot the loss

```

```

20     # list all data in history
21     print(history.history.keys())
22     # summarize history for loss
23     plt.plot(history.history['acc'])
24     plt.plot(history.history['val_acc'])
25     plt.title('model acc')
26     plt.ylabel('acc')
27     plt.xlabel('epoch')
28     plt.legend(['acc', 'val_acc'], loc='upper left')
29     plt.show()
30
31     else:
32         classifier.fit(feature_vector_train, label)
33
34     # predict the labels on validation dataset
35     predictions = classifier.predict(feature_vector_test)
36     predictions = np rint(predictions)
37
38     num_classes = 2
39     return (metrics.accuracy_score(predictions, test_y),
40            tf.confusion_matrix(predictions, test_y, num_classes))

```

Foram treinados 29 classificadores diferentes usando os cinco possíveis atributos. Para os algoritmos clássicos de aprendizado de máquina, foram usados como entrada os vetores *CountVector* e os três vetores TF-IDF. Já para os algoritmos de *deep learning*, foi usado o *Word2Vec* (Tabela 5.1). O melhor classificador encontrado foi o *Suport Vector Machine* com acurácia de 92,80%.

Tabela 5.1: Acurácia dos modelos de classificação

Classificador	Count Vector	TFIDF Palavra	TFIDF Ngram	TFIDF Caractere	Word2Vec
Naive Bayes	0,7405	0,7275	0,8926	0,8360	
Logistic Regression	0,7428	0,7417	0,9198	0,7971	
Suport Vector Machine	0,7535	0,7452	<b>0,9280</b>	0,8478	
Random Forest	0,6898	0,7004	0,8561	0,8042	
Xtereme Gradient Boosting	0,7358	0,7476	0,9221	0,8867	
Multi Layer Perceptron	0,7665	0,7346	0,9186	0,8832	
Convolutional NN					0,8620
Recurrent NN - LSTM					0,8820
Recurrent NN - GRU					0,8726
Bidirectional Recurrent NN					0,8620
Recurrent Convolutional NN					0,8679

### 5.3

#### Teses relevantes para o domínio óleo e gás

Na etapa anterior, foram treinados modelos capazes de classificar se um texto é relevante ou não para o domínio de óleo e gás. Com esses classificadores em mãos, foi possível ler centenas de milhares de documentos de diversas instituições e decidir quais deveriam compor o corpus de óleo

e gás. Primeiramente, foi utilizado o mesmo *crawler* da BDTD para ler todos os resumos de teses e dissertações. Para aqueles documentos que se mostrassem relevantes, foram construídos novos *crawlers* para os repositórios das instituições de pesquisa para fazer o *download* dos arquivos.

A busca pelas teses relevantes foi feita uma instituição por vez. Por exemplo, inicialmente buscou-se na BDTD todas os documentos da UFBA<sup>12</sup>. Foram recuperados quase 10.000 *links* com resumos de documentos dessa instituição. Esses resumos tiveram que ser pré-processados da mesma maneira que foram processados os textos usados no treinamento. Cada um desses resumos foi passado pelo classificador que fez uma predição quanto a relevância ao domínio de petróleo. O modelo classificou cada texto com um valor entre 0 a 1, onde 0 eram os mais relevantes para o domínio do petróleo. Teoricamente, poderiam ser recuperados todos os documentos com predição abaixo de 0,5. No entanto, buscando ser conservador e ter certeza que os documentos recuperados pertenciam ao domínio de interesse, foram recuperados apenas aqueles com predição abaixo de 0,2. Usando esse procedimento para todas as 15 instituições foram recuperados 4.467 teses e dissertações (Tabela 5.2). Ou seja, de todos os documentos disponíveis na BDTD, apenas uma pequena fração era relevante para fazer parte do corpus de óleo e gás.

Tabela 5.2: Documentos recuperados por instituição

	Total de Teses	Teses recuperadas	Percentual de teses recuperadas sobre o total
<b>Total</b>	<b>358,942</b>	<b>4,467</b>	<b>1.24%</b>
USP	66,077	685	1.04%
UNICAMP	45,644	935	2.05%
UNESP	38,008	291	0.77%
UFSC	36,971	308	0.83%
UFRGS	35,665	471	1.32%
UFV	23,446	44	0.19%
UFMG	22,060	146	0.66%
UNB	20,780	29	0.14%
UFPE	20,551	386	1.88%
UFRN	12,625	587	4.65%
UFSCAR	11,076	97	0.88%
UFBA	9,916	186	1.88%
UFES	9,720	204	2.10%
UFS	4,806	71	1.48%
UFF	1,597	27	1.69%

## 5.4

### Crawler para os repositórios institucionais

Uma vez identificados os documentos relevantes para a montagem do corpus, foi necessário recuperar os metadados presentes na BDTD e o arquivo com o texto integral depositado nos repositórios de cada instituição. Os

<sup>12</sup>Universidade Federal da Bahia

```
{
  "http://repositorio.ufba.br/ri/handle/ri/15263": {
    "Title": "A influência dos fatores humanos nas técnicas de análise de risco APP e APR: estudo de caso de uma plataforma de perfuração de petróleo no Nordeste ",
    "Nível de Acesso": "OpenAccess",
    "Data de Defesa": "2013",
    "Autor/a/s": "\n\n Barroso, Marise Paixão",
    "Orientador/a/s": "\n\n Melo, Silvio Alexandre Beisl Vieira de",
    "Co-orientador/a/s": "\n\n Ávila Filho, Salvador",
    "Tipo Documento": "Dissertação",
    "Idioma": "por",
    "Instituição de Defesa": "\nUniversidade Federal da Bahia. Escola Politécnica\n\n",
    "Programa": "\nEngenharia Industrial\n\n",
    "Assuntos em Português": [
      "Avaliação de riscos",
      "Antropologia empresarial",
      "Plataformas de perfuração"
    ],
    "Áreas de Conhecimento": null,
    "Download Texto Completo": "http://repositorio.ufba.br/ri/handle/ri/15263",
    "Resumo Português": "Este trabalho estuda a influência dos fatores humanos nas técnicas de Análise Preliminar de Perigo e Análise Preliminar de Risco aplicadas ao estudo de caso de uma plataforma de perfuração autoelevável. A fundamentação teórica do tema foi realizada através de uma revisão da literatura científica pertinente, que permitiu não só a revisão dos conceitos básicos bem como sua aplicação no gerenciamento de riscos, erro humano e suas causas. De acordo com o quadro teórico pesquisado, as informações aqui consolidadas e aplicadas propõem avaliar os fatores humanos nos acidentes. As técnicas de Análise Preliminar de Perigo Sócio-Humana e Análise Preliminar de Risco Sócio-Humana foram desenvolvidas objetivando identificar problemas sócio-humanos e contribuir para uma discussão sobre a forma de analisar os riscos oriundos dos conflitos comportamentais e dos impactos causados tanto nas comunidades internas como externas, do ambiente de plataforma. As análises foram resultantes de sessões de APRSH que pretendem mitigar conflitos que implicam em acidentes, tendo como referência os maiores índices de ocorrências registradas no ano de 2011. Nesta apuração buscou-se identificar as causas-raízes e embates dos acidentes nos vizinhos, na natureza e na sociedade. Os dados examinados foram coletados por meio de entrevistas individuais e extraiados de trabalhos de campo realizado por um grupo de especialistas envolvidos neste caso. São analisados ainda os fatores operacionais necessários para manter o regime visando à melhoria da saúde, o bem-estar, a segurança, o conforto e a produtividade. Os resultados aqui apresentados indicam os benefícios de se prevenir um possível conflito comportamental (interno e/ou externo) e os respectivos impactos nos trabalhadores e nas partes interessadas do ramo de plataformas offshore.",
    "Resumo Inglês": "This work studies the influence of human factors in risk analysis techniques applied to the case study of an offshore oil platform. This theme has been critically reviewed from the state of art, including risk management, human errors and their causes. The purpose is to assess how the human factors affect the accidents. Techniques APRSH and APRSH were applied to the case study aiming at the identification of social and humans factors and to assess risks from social conflicts and their impacts on both internal and external communities. Assessment was performed from the APRSH sessions whose objective was to mitigate conflicts due to accidents, taking in account the highest incidents rates recorded in 2011. Also the root causes of accidents were identified and their impact on the neighborhood, on the nature and on the society. Data were obtained by individual interviews as well as from fieldwork carried out by a group of experts. This dissertation also assessed the occupational factors needed to keep fulfill the health, welfare, safety, comfort and productivity requirements of the oil business. The results indicate the benefits of foreseeing a possible social conflict (internal and / or external) and their impacts on workers and stakeholders in the field of offshore platforms.",
    "Classificador": "0.117917411",
    "Banca": "\n\n Gonçalves Filho, Anastácio Pinto, \n\n Pontes Junior, Gerardo Portela da",
    "PDF_ID": "UFBA_15263"
  },
}
```

Figura 5.5: Exemplo de metadados de uma dissertação gravado em formato JSON. (Fonte: Elaborada pelo autor)

The screenshot shows the UFBA repository interface. On the left, a sidebar lists navigation options like 'Início', 'Sobre', 'Contato', etc. The main content area displays the document details, including the title, author, and a summary. On the right, the HTML source code of the page is visible, with a red box highlighting the tag `<a target='_blank' href='\"http://repositorio.ufba.br/ri/handle/ri/15263/VerArquivo?file=VerArquivo\"'>VerArquivo</a>`.

Figura 5.6: Identificando *tag* para recuperar documento no repositório institucional da UFBA[7]

metadados foram agrupados em um dicionário e gravados em um arquivo em formato JSON (Figura 5.5).

Já para a recuperação dos arquivos completos, foram criados *crawlers* para os repositórios de cada uma das 15 instituições. O *link* para o documento completo é um dos metadados presentes na BDTD. No entanto, para cada repositório, foi necessário identificar a *tag html* específica que permitiria acessar automaticamente o arquivo. No caso da UFBA, por exemplo, o arquivo poderia ser encontrado procurando pelo texto “View/Open” em todas as *tags* “a” (Figura 5.6). Desta forma, foi possível recuperar automaticamente todos os 186 documentos desta instituição.

---

**Código 5:** *Crawler* para recuperar todas os documentos do repositório institucional da UFBA

---

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 for tese in metadados_ufba.iterrows():
8     print(tese[1]['PDF_ID'])
9     try:
10         #preparar a url
11         url = tese[1]['Download Texto Completo:']
12
13         #Fazer requisição e parsear o arquivo html
14         f = requests.get(url, proxies = proxies).text
15         soup = bs(f, "html.parser")
16
17         #Coletando link para arquivo das teses
18         links = []
19         for doc in soup.find_all('a', href=True):
20             if doc.get_text() == 'View/Open':
21                 links.append(doc['href'])
22
23         #Recuperando e gravando arquivo PDF
24         url = 'http://repositorio.ufba.br' + links[0]
25         pdf = requests.get(url, proxies = proxies)
26         filename = tese[1]['PDF_ID'] + '.pdf'
27         with open(filename, 'wb') as f:
28             f.write(pdf.content)
29     except:
30         pass
```

---

## 5.5

### Extração de informações dos documentos

Uma vez que os documentos relevantes estavam disponíveis, a próxima etapa foi efetivamente montar um corpus de óleo e gás. Para isso, foi necessário extrair os textos dos documentos em formato PDF. A solução utilizada para a extração dos textos dos arquivos foi a ferramenta Apache Tika<sup>13</sup>. Essa é uma das ferramentas mais populares para extração de texto, pois ela suporta diversos formatos de arquivos, inclusive PDF. Muitas aplicações, como diversos motores de busca, utilizam o Apache Tika como solução de extração de texto.

No entanto, apesar de ser muito popular, o texto extraído utilizando o Apache Tika ainda apresenta algumas inconsistências. Por exemplo, textos

<sup>13</sup><https://tika.apache.org/>

oriundos de tabelas são convertidos como textos comuns e misturados com o restante do texto. O mesmo ocorre com legendas e textos presentes dentro das figuras. Desta forma, o texto final que irá compor o Petrolês poderá conter algum ruído indesejado.

Para mitigar esses problemas, a próxima etapa deste trabalho já está sendo planejada. Uma segunda extração do texto será realizada utilizando uma ferramenta de extração própria, que está em desenvolvimento. Essa ferramenta usará técnicas de visão computacional, mais especificamente redes neurais convolucionais, para identificar as partes do documento onde há apenas texto. Com isso, será evitado a extração de elementos irrelevantes dos documentos, como sumários, referências bibliográficas, tabelas, figuras e fórmulas.

Os textos extraídos dessas 4.302 teses e dissertações compuseram o corpus final de óleo e gás. Por fim, o Petrolês possui 74.445.798 palavras em 5.999.496 sentenças, sendo o maior corpus do domínio de óleo e gás na língua portuguesa.



## 6

### Conclusão

Neste trabalho foi criado o Petrolês, um corpus de óleo e gás em português para ser utilizado nas inúmeras tarefas de processamento de linguagem natural. Esse corpus é o maior conjunto de documentos disponível em língua portuguesa especializado na indústria do petróleo. A vantagem de se ter um corpus em um domínio específico é a possibilidade de se capturar jargões técnicos e usos específicos de palavras que dificilmente seria encontrado em textos de domínio geral.

Para a montagem deste corpus, foi necessário criar dois tipos de robôs de extração de informações de páginas da internet, os chamados *crawlers*. O primeiro buscou informações no repositório da Base Digital de Teses e Dissertações do IBICT. Essa base possui informações sobre as principais instituições de pesquisas brasileiras. O segundo tipo de *crawler* recuperou os arquivos completos das teses e dissertações nos repositórios de cada instituição.

No entanto, de todas as teses e dissertações produzidas, apenas 1,24% são relevantes para a indústria de óleo e gás. Para diferenciar os documentos relevantes dos não relevantes, foi treinado um modelo baseado em aprendizado de máquina para fazer a classificação dos documentos.

Por fim, foi possível ler automaticamente o resumo de mais de 350 mil teses e dissertações e baixar apenas as 4.302 mais relevantes. Esses documentos serviram de base para um corpus com quase 75 milhões de palavras e que servirá de referência para os futuros trabalhos de processamento de linguagem natural na indústria do petróleo.

## Bibliografia

- [1] Charu C. Aggarwal, ed. *Data classification: algorithms and applications*. Chapman & Hall/CRC data mining and knowledge discovery series 35. OCLC: 904721748. Boca Raton, Fla.: CRC Press/Chapman & Hall, 2014. 671 pp. ISBN: 978-1-4665-8675-8 978-1-4665-8674-1.
- [2] Meiryum Al. *The Best 25 Datasets for Natural Language Processing*. lionbridge.ai. 9 de jul. de 2019. URL: <https://lionbridge.ai/datasets/the-best-25-datasets-for-natural-language-processing/> (acesso em 06/12/2019).
- [3] ANP. *Instituições credenciadas*. anp.gov.br. 8 de out. de 2019. URL: <http://www.anp.gov.br/pesquisa-desenvolvimento-e-inovacao/credenciamentos-de-instituicoes/instituicoes-credenciadas> (acesso em 10/12/2019).
- [4] Gustavo Vianna Avila. “Análise de sentimento para textos curtos”. Tese de dout. Rio de Janeiro: FGV EMAP, 2017.
- [5] R. Baeza-Yates e B. Ribeiro-Neto. *Recuperação de Informação - 2ed: Conceitos e Tecnologia das Máquinas de Busca*. Bookman Editora, 2013. ISBN: 978-85-8260-049-8. URL: <https://books.google.com.br/books?id=YWk3AgAAQBAJ>.
- [6] Shivam Bansal. *A Comprehensive Guide to Understand and Implement Text Classification in Python*. Analytics Vidhya. 23 de abr. de 2018. URL: <https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/> (acesso em 14/08/2019).
- [7] Marise Paixão Barroso. *A influência dos fatores humanos nas técnicas de análise de risco APP e APR: estudo de caso de uma plataforma de perfuração de petróleo no Nordeste*. REPOSITÓRIO Institucional UFBA. URL: <https://repositorio.ufba.br/ri/handle/ri/15263> (acesso em 29/12/2019).
- [8] J. Brownlee. *Deep Learning for Natural Language Processing: Develop Deep Learning Models for your Natural Language Problems*. Machine Learning Mastery, 2017. URL: [https://books.google.com.br/books?id=\\_pmoDwAAQBAJ](https://books.google.com.br/books?id=_pmoDwAAQBAJ).

- [9] LEXICAL COMPUTING. *Corpus types: monolingual, parallel, multilingual*. sketchengine.eu. URL: <https://www.sketchengine.eu/corpora-and-languages/corpus-types/> (acesso em 05/12/2019).
- [10] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. Em: *arXiv:1810.04805 [cs]* (24 de mai. de 2019). arXiv: 1810.04805. URL: <http://arxiv.org/abs/1810.04805> (acesso em 09/12/2019).
- [11] IBICT. *Indicadores da Biblioteca Digital Brasileira de Teses e Dissertações (BDTD)*. 2019. URL: <http://bdttd.ibict.br/vufind/Content/statics> (acesso em 10/12/2019).
- [12] IBICT. *Instituto Brasileiro de Informação em Ciência e Tecnologia*. Instituto Brasileiro de Informação em Ciência e Tecnologia. 15 de out. de 2018. URL: <http://www.ibict.br/sobre-o-ibict/historico>.
- [13] Masoud Makrehchi e Mohamed S. Kamel. “Automatic Extraction of Domain-Specific Stopwords from Labeled Documents”. Em: *Advances in Information Retrieval*. Ed. por Craig Macdonald et al. Vol. 4956. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 222–233. ISBN: 978-3-540-78645-0 978-3-540-78646-7. DOI: 10.1007/978-3-540-78646-7\_22. URL: [http://link.springer.com/10.1007/978-3-540-78646-7\\_22](http://link.springer.com/10.1007/978-3-540-78646-7_22) (acesso em 20/12/2019).
- [14] Jacopo Mangiavacchi. *Core ML with GloVe Word Embedding and Recursive Neural Network — part 2. Implementing a Natural Language Classifier in iOS with Keras/TensorFlow + Core ML — part 2*. Heartbeat. 25 de abr. de 2018. URL: <https://heartbeat.fritz.ai/coreml-with-glove-word-embedding-and-recursive-neural-network-part-2-d72c1a66b028> (acesso em 20/12/2019).
- [15] Tom McArthur. *The Oxford Companion to the English*. 1992<sup>a</sup> ed. Oxford & New York: Oxford University Press.
- [16] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. Em: *arXiv:1301.3781 [cs]* (6 de set. de 2013). arXiv: 1301.3781. URL: <http://arxiv.org/abs/1301.3781> (acesso em 09/12/2019).
- [17] Alex Nguyen. *12 Best Portuguese Language Datasets for Machine Learning*. lionbridge.ai. 19 de set. de 2019. URL: <https://lionbridge.ai/datasets/best-portuguese-language-datasets-for-machine-learning/> (acesso em 07/12/2019).

- [18] Sk Md Obaidullah et al., ed. *Document processing using machine learning*. 1<sup>a</sup> ed. Boca Raton: CRC Press, 2019. ISBN: 978-0-367-21847-8.
- [19] Arvind Padmanabhan. *Text Corpus for NLP*. devopedia. 28 de out. de 2019. URL: <https://devopedia.org/text-corpus-for-nlp> (acesso em 05/12/2019).
- [20] Guilherme Franco Silva Pinto. “Comportamento informacional e mineração textual no Twitter”. Tese de dout. Universidade Federal de São Carlos, 2018. URL: <https://repositorio.ufscar.br/handle/ufscar/10535>.
- [21] Sudharsan Ravichandiran. *Hands-on deep learning algorithms with python: master deep learning algorithms with extensive math by implementing them using tensorflow*. OCLC: 1124727962. 2019. ISBN: 978-1-78934-451-6. URL: <https://ebookcentral.proquest.com/lib/ubstgallen-ebooks/detail.action?docID=5841215> (acesso em 14/12/2019).
- [22] Fang Yuan, Bo Liu e Ge Yu. “A Study on Information Extraction from PDF Files”. Em: *Advances in Machine Learning and Cybernetics*. Ed. por Daniel S. Yeung et al. Red. por David Hutchison et al. Vol. 3930. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 258–267. ISBN: 978-3-540-33584-9 978-3-540-33585-6. DOI: 10.1007/11739685\_27. URL: [http://link.springer.com/10.1007/11739685\\_27](http://link.springer.com/10.1007/11739685_27) (acesso em 28/12/2019).

## Apêndice 1 - Notebook webscraping BDTD

Script para buscar resumos das teses e dissertações na BDTD

```
[27]: import requests
      from bs4 import BeautifulSoup as bs
      import pandas as pd
      import json

[28]: # Definindo configurações globais de proxy para realizar a extração dentro da
      ↳ rede Petrobras
chave = 'XXXX'
pwd = 'XXXXXXXXX'
proxy_url = 'http://' + chave + ':' + pwd + '@inet-sys.gnet.petrobras.com.br:804/'
proxies = {
    'http' : proxy_url ,
    'https' : proxy_url ,
}

[29]: areas = pd.read_csv('Areas_CNPQ.csv', sep=';')

[30]: #função para coletar link para cada tese de uma determinada áreas do
      ↳ conhecimento do CNPQ

def link_area(area, page):
    #Separar nome e sobrenome do autor
    nivel_1 = area[0]
    nivel_2 = area[1]

    #Preparar nome e sobrenome para a url
    nivel_1 = '+'.join(nivel_1.split())

    if str(nivel_2) != 'nan':
        nivel_2 = '+'.join(nivel_2.split())

    #preparar a url
    url = ('http://bdtd.ibict.br/vufind/Search/Results?filter%5B%5D=dc.
↳subject.cnpq.fl_str_mv%3A"CNPQ%3A%3A' +
        nivel_1 +
        '%3A%3A' +
        nivel_2 +
        '&page=' +
```

```

        str(page))
    else:
        url = ('http://bdttd.ibict.br/vufind/Search/Results?filter%5B%5D=dc.
↪subject.cnpq.fl_str_mv%3A"CNPQ%3A%3A' +
            nivel_1 +
            '&page=' +
            str(page))

    #Fazer requisição e parsear o arquivo html
    f = requests.get(url, proxies = proxies).text
    soup = bs(f, "html.parser")

    #Coletando link para as teses
    links = []
    for doc in soup.find_all('a', href=True):
        if 'title' in doc.get('class', []):
            links.append(doc['href'])
    return links

```

[31]: *#Coletar o link com as teses das áreas não relacionadas à Petrobras*  
 area\_oposta = areas[areas['PETROBRAS'] == 0]

```

n_pages = 10 # Cada página retorna 20 teses
links_area_oposta = []

for area in area_oposta.iterrows():
    for p in range(n_pages):
        link = link_area(area[1], p)
        if link != []:
            links_area_oposta = links_area_oposta + link
        else:
            break

```

[32]: *#Coletar o link com as teses das áreas não relacionadas à Petrobras*  
 mesma\_area = areas[areas['PETROBRAS'] == 1]

```

n_pages = 10 # Cada página retorna 20 teses
links_mesma_area = []

for area in mesma_area.iterrows():
    for p in range(n_pages):
        link = link_area(area[1], p)
        if link != []:
            links_mesma_area = links_mesma_area + link
        else:
            break

```

```
[33]: # Função para coletar os metadados de uma tese ou dissertação da BDTD dado uma URL
def tese_link(url):
    #definir url
    url = 'http://bdtb.ibict.br' + link

    #Requisitar html e fazer o parser
    f = requests.get(url, proxies = proxies).text
    soup = bs(f, "html.parser")

    #Dicionário para armazenar as informações da tese
    tese = {}

    #Adicionar título
    tese['Title'] = soup.find('h3').get_text()
    for doc in soup.find_all('tr'):

        #Identificar atributo
        try:
            atributo = doc.find('th').get_text()
        except:
            pass


        #Verificar se o atributo possui mais de um dado
        for row in doc.find_all('td'):

            #Adicionar o atributo no dicionário
            if row.find('div') == None:
                try:
                    tese[atributo] = doc.find('td').get_text()
                except:
                    pass
            else:
                element = []


                #No dicionário, adicionar todos os dados ao seu respectivo atributo
                for e in doc.find_all('div'):
                    try:
                        element.append(e.find('a').get_text())
                    except:
                        pass
                tese[atributo] = element

    return(tese)
```

```
[38]: teses = {}  
n = 0  
for link in links_area_oposta:  
    url = 'http://bdtd.ibict.br'+link  
    teses[url] = tese_link(link)  
    n = n + 1  
    if n % 500 == 0:  
        print(n)  
        with open('teses_areas_opostas_Large.json', 'w') as fp:  
            json.dump(teses, fp)
```

500   1000   1500   2000   2500   3000   3500   4000   4500   5000   5500   6000     
→6500

```
[39]: teses = {}  
n = 0  
for link in links_mesma_area:  
    url = 'http://bdtd.ibict.br'+link  
    teses[url] = tese_link(link)  
    n = n + 1  
    if n % 500 == 0:  
        print(n)  
        with open('teses_mesmas_areas_Large.json', 'w') as fp:  
            json.dump(teses, fp)
```

500   1000   1500   2000   2500   3000   3500   4000   4500   5000   5500   6000     
→6500



## Apêndice 2 - Notebook classificação de texto

### 1 Classificação de Texto do domínio de óleo e gás

O objetivo deste notebook é fazer um estudo sobre a classificação de textos. Serão usadas diversas técnicas para criar um modelo que classifique resumos de teses de doutorado e dissertação de mestrado. Serão usados documentos elaborados por técnicos da Petrobras, e da Biblioteca Digital de Teses e Dissertações. Esperamos que os modelos classifiquem corretamente os documentos nos seus respectivos domínios.

Baseado no post de Shivam Bansal

Shivam Bansal. **A Comprehensive Guide to Understand and Implement Text Classification in Python**. Analytics Vidhya. 23 de abr. de 2018. url: <https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/> (acesso em 14/08/2019)

```
[1]: # Importando bibliotecas
import warnings
warnings.filterwarnings("ignore")
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn import decomposition, ensemble
from random import shuffle
import pandas as pd
import numpy as np
import xgboost, textblob, string
from keras.preprocessing import text, sequence
import tensorflow as tf
from keras.models import Sequential
from keras.utils import to_categorical
from keras import layers, models, optimizers
from keras.callbacks import EarlyStopping
from keras.layers import Concatenate
from keras.layers import Layer
from keras.layers import Flatten
import tensorflow as tf
from keras.initializers import get
from bs4 import BeautifulSoup as bs
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
import matplotlib.pyplot as plt
import gensim
from gensim.models import Word2Vec
from langdetect import detect
from langdetect import detect_langs
```

Using TensorFlow backend.

## 2 Preparando os dados

Lendo arquivos JSON com os dados das teses Petrobras no BDTD, das teses no BDTD com assunto “Petroleo” e teses de assunto opostos ao de interesse Petrobras (“Linguas, Letras e Artes”, “Arqueologia”, “Demografia”, ...)

```
[2]: teses_Subject_petroleo = pd.read_json('BDTD/New_Subject_petroleo.json', orient = 'index')
teses_petrobras_BDTD = pd.read_json('Petrobras/New_teses_petrobras_BDTD.json', orient = 'index')
tese_mesma_area_Large = pd.read_json('BDTD/teses_mesmas_areas_Large.json', orient = 'index')
teses_areas_opostas_Large = pd.read_json('BDTD/teses_areas_opostas_Large.json', orient = 'index')
```

```
[3]: # Número total de documentos
len(tese_mesma_area_Large) + len(teses_areas_opostas_Large)
```

```
[3]: 11532
```

```
[4]: # Unindo as teses de Petróleo
teses_petroleo = teses_Subject_petroleo
teses_petroleo = teses_petroleo.append(teses_petrobras_BDTD)
# Excluindo teses duplicadas
teses_petroleo = teses_petroleo[~teses_petroleo.index.duplicated(keep='first')]
```

```
[5]: # Unindo as teses de todas as áreas
teses_areas = tese_mesma_area_Large
teses_areas = teses_areas.append(teses_areas_opostas_Large)
# Excluindo teses duplicadas
teses_areas = teses_areas[~teses_areas.index.duplicated(keep='first')]
```

Acrescentando a classe nos dois DataFrame

```
[6]: teses_petroleo['classe'] = 'Petroleo'
teses_areas ['classe'] = 'Todas Areas'
```

Verificando a existência de teses duplicadas nas duas classes

```
[7]: # Unindo as duas classes de documentos
todos = teses_petroleo
todos = todos.append(teses_areas)
len(todos)
```

[7]: 13885

```
[16]: # Verificando a existência de documentos duplicados
todos = todos[~todos.index.duplicated(keep='first')]
len(todos)
```

[16]: 13845

```
[8]: #SEparando novamente
teses_petroleo = todos[todos['classe'] == 'Petroleo']
teses_areas = todos[todos['classe'] == 'Todas Areas']
```

Verificando balanceamento das duas classes

```
[9]: print ('teses_petroleo: ', len(teses_petroleo))
print ('teses_areas: ', len(teses_areas))
```

teses\_petroleo: 2366

teses\_areas: 11519

Verificando os campos de resumo

```
[10]: # Excluindo documentos sem resumo em português
teses_petroleo = teses_petroleo[(teses_petroleo['Resumo Português:'].notnull())]
teses_areas = teses_areas[(teses_areas['Resumo Português:'].notnull())]
```

```
[11]: # Função que recebe um texto e separa a parte português da parte em inglês
def separacao_port_engl(abstract):

    # Tokeniza os resumos em sentenças
    mix_sent = nltk.sent_tokenize(abstract)

    # Algumas sentenças vem unidas sem espaço.
    # Portanto é necessário encontra o ponto final para quebrar a sentença em
    ↪ duas.
    new_mix = []
    for sent in mix_sent:
        position = sent.find('.')
        if position != len(sent)-1:
            sent_1 = sent[:position+1]
            sent_2 = sent[position+1:]
            new_mix.append(sent_1)
            new_mix.append(sent_2)
        else:
            new_mix.append(sent)

    mix_sent = new_mix

    # Para cada sentença, identificar se ela está em português ou inglês
    port = []
    engl = []
```

```

for sent in mix_sent:
    try:
        if detect (sent) == 'pt':
            port.append(sent)
        else:
            engl.append (sent)
    except:
        pass

# As sentenças são unidas novamente
port = " ".join(port)
engl = " ".join(engl)

# A função retorna os resumos em cada idioma
return(port, engl)

```

```

[12]: # Separando português e inglês para teses petróleo
columns_pt = teses_petroleo['Resumo Português:'].apply(lambda x:↵
↵separacao_port_engl(x)[0])
columns_en = teses_petroleo['Resumo Português:'].apply(lambda x:↵
↵separacao_port_engl(x)[1])
teses_petroleo['Resumo Português:'] = columns_pt
teses_petroleo['Resumo inglês 2:'] = columns_en

# Separando português e inglês para teses das demais áreas
columns_pt = teses_areas['Resumo Português:'].apply(lambda x:↵
↵separacao_port_engl(x)[0])
columns_en = teses_areas['Resumo Português:'].apply(lambda x:↵
↵separacao_port_engl(x)[1])
teses_areas['Resumo Português:'] = columns_pt
teses_areas['Resumo inglês 2:'] = columns_en

```

Excluindo novamente as teses sem resumo em portugues

```

[13]: teses_petroleo = teses_petroleo[(teses_petroleo['Resumo Português:'].notnull())]
teses_areas = teses_areas[(teses_areas['Resumo Português:'].notnull())]

```

Preprocessando o texto e retirando stopwords

```

[14]: # Letras em minúsculas
teses_petroleo['Resumo Português:'] = teses_petroleo['Resumo Português:'].str.
↵lower()
teses_areas['Resumo Português:'] = teses_areas['Resumo Português:'].str.lower()

```

```

[16]: # Preprocessando os textos
teses_petroleo['Resumo Português:'] = (teses_petroleo['Resumo Português:']
↵.apply(gensim.utils.simple_preprocess)
↵.str.join(" "))
teses_areas['Resumo Português:'] = (teses_areas['Resumo Português:']
↵.apply(gensim.utils.simple_preprocess)

```

```
.str.join(" ")
```

```
[17]: # Importando as bibliotecas de stopwords
      nltk.download('stopwords')

      # Mapeando stopwords com NLTK
      stopwordsIngles = stopwords.words("portuguese")

      def remove_stopwords(abstract):
          without_stopwords = []
          for word in abstract:
              if word not in stopwordsIngles:
                  without_stopwords.append(word)
          return(without_stopwords)

      # Excluindo stopwords
      teses_petroleo['Resumo Português:'] = teses_petroleo['Resumo Português:'].
      ↪ apply(remove_stopwords)
      teses_areas['Resumo Português:'] = teses_areas['Resumo Português:'].
      ↪ apply(remove_stopwords)

      # Unindo novamente o texto em uma única string
      teses_petroleo['Resumo Português:'] = teses_petroleo['Resumo Português:'].str.
      ↪ join(" ")
      teses_areas['Resumo Português:'] = teses_areas['Resumo Português:'].str.join(" ")

[2]: # Gravando textos preprocessados em um arquivo JSON
      #teses_petroleo.to_json('BDTD/tese_petroleo_processada.json', orient = 'index')
      #teses_areas.to_json('BDTD/tese_areas_processada.json', orient = 'index')
      # lendo textos preprocessados de arquivos JSON
      teses_petroleo = pd.read_json('BDTD/tese_petroleo_processada.json', orient =
      ↪ 'index')
      teses_areas = pd.read_json('BDTD/tese_areas_processada.json', orient = 'index')
```

## 2.1 Dividindo o conjunto de treino, validação e de teste

Vamos dividir os dados em 80% treino e 20% teste

```
[3]: #Função que recebe um dataframe com as teses e retorna dois dataframes com dados
      ↪ de treino e teste.
      # 'train' é a fração dos dados para treino, o restante é para teste
      def train_test(teses, train):
          corte_train = int(round((len(teses)*train),0))
          teses = teses.sample(frac=1)
          teses_train = teses[:corte_train]
          teses_test = teses[corte_train:]
          return(teses_train, teses_test)
```

```
[4]: # Os documentos são balanceados para ficarem com a mesma quantidade
teses_areas = teses_areas.sample(len(teses_petroleo))
```

```
[5]: print(len(teses_areas))
      print(len(teses_petroleo))
```

```
2121
```

```
2121
```

```
[6]: # São separadas as frações para treino e teste
teses_petroleo_train, teses_petroleo_test = train_test(teses_petroleo, 0.8)
teses_areas_train, teses_areas_test = train_test(teses_areas, 0.8)
```

```
[7]: # Os dados de treino e teste são unidos e embaralhados
# Train
tese_train = teses_petroleo_train
tese_train = tese_train.append(teses_areas_train)
tese_train = tese_train.sample(frac=1).reset_index(drop=True)

#Test
tese_test = teses_petroleo_test
tese_test = tese_test.append(teses_areas_test)
tese_test = tese_test.sample(frac=1).reset_index(drop=True)
```

```
[8]: # Separando apenas os texto e classes para treinar os classificadores
train_x = tese_train['Resumo Português:']
train_y = tese_train['classe']
```

```
test_x = tese_test['Resumo Português:']
test_y = tese_test['classe']
```

```
[9]: # Codificando as classes para as variáveis 0 e 1
encoder = preprocessing.LabelEncoder()
train_y = encoder.fit_transform(train_y)
test_y = encoder.transform(test_y)
```

```
[10]: print ('Petrobras = ', encoder.transform(['Petroleo'])[0])
      print ('Outro = ', encoder.transform(['Todas Areas'])[0])
```

```
Petrobras = 0
```

```
Outro = 1
```

### 3 Feature Engineering

O próximo passo é criar os atributos dos textos. Nesta etapa o texto bruto será transformado em vetores e novos atributos serão criados a partir dos dados atuais.

#### Count Vectors

Count Vector é uma notação de matriz, onde cada linha representa um documento, cada coluna

representa um termo do corpus, e cada célula representa a frequência de um determinado termo em um documento em particular.

```
[11]: # Criando um objeto Count Vector
count_vect = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}')
```

```
count_vect.fit(tese_train['Resumo Português:'])
```

```
# Transforma os dados de treino e teste usando o objeto Count Vector
xtrain_count = count_vect.transform(train_x)
xtest_count = count_vect.transform(test_x)
```

### TF-IDF Vectors

TF-IDF score representa a importância relativa dos termos em um documento e no corpus inteiro. TF-IDF score é composto por:

$TF(t) = (\text{Número de vezes que o termo } t \text{ aparece em um documento}) / (\text{Número total de termos em um documento})$   
 $IDF(t) = \log_e(\text{Número total de documentos} / \text{Número de documentos que contém o termo } t)$

Os vetores TF-IDF podem ser gerados com diferentes níveis de tokens (palavras, caracteres, n-grams)

```
[12]: # word level tf-idf
tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}',
    ↪max_features=5000)
tfidf_vect.fit(tese_train['Resumo Português:'])
xtrain_tfidf = tfidf_vect.transform(train_x)
xtest_tfidf = tfidf_vect.transform(test_x)
```

```
# ngram level tf-idf
tfidf_vect_ngram = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}',
    ↪ngram_range=(2,3), max_features=5000)
tfidf_vect_ngram.fit(tese_train['Resumo Português:'])
xtrain_tfidf_ngram = tfidf_vect_ngram.transform(train_x)
xtest_tfidf_ngram = tfidf_vect_ngram.transform(test_x)
```

```
# characters level tf-idf
tfidf_vect_ngram_chars = TfidfVectorizer(analyzer='char',
    ↪token_pattern=r'\w{1,}', ngram_range=(2,3), max_features=5000)
tfidf_vect_ngram_chars.fit(tese_train['Resumo Português:'])
xtrain_tfidf_ngram_chars = tfidf_vect_ngram_chars.transform(train_x)
xtest_tfidf_ngram_chars = tfidf_vect_ngram_chars.transform(test_x)
```

### Word Embeddings

Word embeddings é uma forma de representação de palavras e documentos usando uma vetores. A posição das palavras em um espaço vetorial é aprendido do texto e é baseado nas palavras que o rodeiam. Word embeddings podem ser treinados usando como input o próprio corpus ou pode ser gerado usando modelos pré treinados como Glove, FastText ou Word2Vec.

#### implementando word2vec

```
[29]: # unindo todos os textos
corpus = todos['Resumo Português:']
```

```
corpus = corpus.str.cat(sep=' ')
```

```
[30]: # Criando uma lista de sentenças e embaralhando-as
corpus = nltk.sent_tokenize(corpus)
shuffle(corpus)
```

```
[31]: # Tokenizando as sentenças
corpus_processado = []
for sentence in corpus:
    corpus_processado.append(word_tokenize(sentence))
```

```
[32]: # Treinando um modelo de Word2Vec
BDTD_word2vec_50 = Word2Vec(corpus_processado, size=50, window=10, min_count=1,
    ↪workers=4, iter=100)
```

```
[33]: # Exemplo do vetor da palavra água
BDTD_word2vec_50.wv['água']
```

```
[33]: array([-3.6960294e+00, -1.3259159e+00, -2.2412670e+00,  5.7865171e+00,
          2.2329526e+00, -5.5173335e+00, -5.9714718e+00,  7.3408980e+00,
          3.9766235e+00, -4.5603027e+00,  4.6718297e+00, -2.3560665e+00,
         -6.8443626e-01, -6.0587273e+00,  3.6310940e+00, -1.5324932e+01,
          2.0639896e+00,  6.0305029e-01,  3.7788615e+00,  4.0281076e+00,
          3.9048851e+00,  1.1163657e+00,  2.3122082e+00,  4.4901333e+00,
         -3.3812339e+00,  2.3412783e+00, -5.2094436e+00, -1.1195867e+00,
         -3.8026874e+00,  9.2307749e+00, -1.8853464e+00, -6.3019433e+00,
          3.5904100e+00, -2.6272061e+00, -4.8584223e+00,  7.7866459e+00,
         -1.3319616e+00,  5.1870914e+00, -5.6637077e+00, -1.5475802e+00,
          9.2607457e-03, -5.3038816e+00,  3.9777195e+00, -6.1717930e+00,
         -5.0633631e+00, -1.7807996e+00,  8.1977360e-03, -3.5111365e+00,
         -5.9699243e-01,  1.4175992e+00], dtype=float32)
```

```
[34]: # Vetores mais similares a palavra água
BDTD_word2vec_50.wv.similar_by_word('água')
```

```
[34]: [('vazão', 0.730722188949585),
      ('solo', 0.6993394494056702),
      ('líquido', 0.687868058681488),
      ('ar', 0.6873413920402527),
      ('areia', 0.6845143437385559),
      ('argila', 0.6819689869880676),
      ('umidade', 0.6753614544868469),
      ('ozônio', 0.67515629529953),
      ('irrigação', 0.6692394018173218),
      ('óleo', 0.6592279076576233)]
```

```
[15]: # Gravando e lendo os modelos de embeddings
#BDTD_word2vec_50.save("Embeddings\BDTD_word2vec_50")
BDTD_word2vec_50 = Word2Vec.load("Embeddings\BDTD_word2vec_50")
```



```
[16]: # Indexando as palavras presentes no modelo Word2Vec
word2index = {}
for index, word in enumerate(BDTD_word2vec_50.wv.index2word):
    word2index[word] = index

[17]: # Indexando as palavras presentes no modelo Word2Vec
word2index = {}
for index, word in enumerate(BDTD_word2vec_50.wv.index2word):
    word2index[word] = index

# Função para indexar o texto usando os índices do modelo Word2Vec
def index_pad_text(text, maxlen, word2index):
    maxlen = 400
    new_text = []
    for sent in text:
        temp_sent = []
        for word in word_tokenize(sent):
            try:
                temp_sent.append(word2index[word])
            except:
                pass
        # Estebelecendo um limite máximo de palavras para cada resumo (padding)
        if len(temp_sent) > maxlen:
            temp_sent = temp_sent[:400]
        else:
            temp_sent += [0] * (maxlen - len(temp_sent))
        new_text.append(temp_sent)

    return np.array(new_text)

maxlen = 400
train_seq_x = index_pad_text(train_x, maxlen, word2index)
test_seq_x = index_pad_text(test_x, maxlen, word2index)
```

## 4 Model Building

A etapa final do framework de classificação de texto é treinar um classificador usando os atributos criados anteriormente. Os seguintes algoritmos de aprendizado de máquina foram implementados:

- Naive Bayes Classifier
- Linear Classifier - Logistic Regression
- Support Vector Machine
- Bagging Models - Random Forest
- Boosting Models - Xtereme Gradient Boosting
- Shallow Neural Networks
- Deep Neural Networks

- Convolutional Neural Network (CNN)
- Long Short Term Modelr (LSTM)
- Gated Recurrent Unit (GRU)
- Bidirectional RNN
- Recurrent Convolutional Neural Network (RCNN)
- Other Variants of Deep Neural Networks

A função abaixo é usada para treinar os modelos. Ela aceita o classificador, o vetor de atributos dos dados de treinamento, as classes de treinamento, os atributos dos dados de teste e a informação se o classificador é uma rede neural. Com essas informações o modelo é treinado, a acurácia pe computada e, nos casos das redes neurais, um gráfico das épocas de treinamento é apresentado.

```
[18]: def train_model(classifier, feature_vector_train, label, feature_vector_test,
    ↪is_neural_net=False):
    # fit the training dataset on the classifier

    if is_neural_net:
        callbacks = EarlyStopping(monitor='val_acc', patience=10,
    ↪restore_best_weights=True)
        history = classifier.fit(feature_vector_train,
                                label, #to_categorical(label),
                                epochs=1000,
                                batch_size=64,
                                validation_split=0.25,
                                callbacks=[callbacks])

    # plot the loss
    # list all data in history
    print(history.history.keys())
    # summarize history for loss
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model acc')
    plt.ylabel('acc')
    plt.xlabel('epoch')
    plt.legend(['acc', 'val_acc'], loc='upper left')
    plt.show()

    else:
        classifier.fit(feature_vector_train, label)

    # predict the labels on validation dataset
    predictions = classifier.predict(feature_vector_test)
    predictions = np rint(predictions)

    num_classes = 2
    return (metrics.accuracy_score(predictions, test_y),
```

```
tf.confusion_matrix(predictions, test_y, num_classes))
```

### Naive Bayes

```
[19]: # Naive Bayes on Count Vectors
accuracy, confusion = train_model(naive_bayes.MultinomialNB(), xtrain_count,
    ↪train_y, xtest_count)
print ("NB, Count Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Naive Bayes on Word Level TF IDF Vectors
accuracy, confusion = train_model(naive_bayes.MultinomialNB(), xtrain_tfidf,
    ↪train_y, xtest_tfidf)
print ("NB, WordLevel TF-IDF Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Naive Bayes on Ngram Level TF IDF Vectors
accuracy, confusion = train_model(naive_bayes.MultinomialNB(),
    ↪xtrain_tfidf_ngram, train_y, xtest_tfidf_ngram)
print ("NB, N-Gram Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Naive Bayes on Character Level TF IDF Vectors
accuracy, confusion = train_model(naive_bayes.MultinomialNB(),
    ↪xtrain_tfidf_ngram_chars, train_y, xtest_tfidf_ngram_chars)
print ("NB, CharLevel Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())
```

```
NB, Count Vectors:  0.7405660377358491
[[324 120]
 [100 304]]
NB, WordLevel TF-IDF Vectors:  0.7275943396226415
[[336 143]
 [ 88 281]]
NB, N-Gram Vectors:  0.8926886792452831
[[387  54]
 [ 37 370]]
NB, CharLevel Vectors:  0.8360849056603774
[[382  97]
 [ 42 327]]
```

### Linear Classifier - Logistic Regression

```
[20]: # Linear Classifier on Count Vectors
```

```

accuracy, confusion = train_model(linear_model.LogisticRegression(),
    ↪xtrain_count, train_y, xtest_count)
print ("LR, Count Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Linear Classifier on Word Level TF IDF Vectors
accuracy, confusion = train_model(linear_model.LogisticRegression(),
    ↪xtrain_tfidf, train_y, xtest_tfidf)
print ("LR, WordLevel TF-IDF Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Linear Classifier on Ngram Level TF IDF Vectors
accuracy, confusion = train_model(linear_model.LogisticRegression(),
    ↪xtrain_tfidf_ngram, train_y, xtest_tfidf_ngram)
print ("LR, N-Gram Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Linear Classifier on Character Level TF IDF Vectors
accuracy, confusion = train_model(linear_model.LogisticRegression(),
    ↪xtrain_tfidf_ngram_chars, train_y, xtest_tfidf_ngram_chars)
print ("LR, CharLevel Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

```

```

LR, Count Vectors:  0.7452830188679245
[[325 117]
 [ 99 307]]
LR, WordLevel TF-IDF Vectors:  0.7417452830188679
[[311 106]
 [113 318]]
LR, N-Gram Vectors:  0.9198113207547169
[[386  30]
 [ 38 394]]
LR, CharLevel Vectors:  0.7971698113207547
[[335  83]
 [ 89 341]]

```

### Support Vector Machine (SVM)

[21]:

```

# SVM on Count Vectors
accuracy, confusion = train_model(svm.SVC(gamma='scale'), xtrain_count, train_y,
    ↪xtest_count)
print ("SVM, Count Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

```

```

# SVM on Word Level TF IDF Vectors
accuracy, confusion = train_model(svm.SVC(gamma='scale'), xtrain_tfidf, train_y,
    ↪xtest_tfidf)
print ("SVM, Word Level TF IDF Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# SVM on Ngram Level TF IDF Vectors
accuracy, confusion = train_model(svm.SVC(gamma='scale'), xtrain_tfidf_ngram,
    ↪train_y, xtest_tfidf_ngram)
print ("SVM, Ngram Level TF IDF Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# SVM on Character Level TF IDF Vectors
accuracy, confusion = train_model(svm.SVC(gamma='scale'),
    ↪xtrain_tfidf_ngram_chars, train_y, xtest_tfidf_ngram_chars)
print ("SVM, Character Level TF IDF Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

```

```

SVM, Count Vectors:  0.7535377358490566
[[327 112]
 [ 97 312]]
SVM, Word Level TF IDF Vectors:  0.7452830188679245
[[311 103]
 [113 321]]
SVM, Ngram Level TF IDF Vectors:  0.9280660377358491
[[388 25]
 [ 36 399]]
SVM, Character Level TF IDF Vectors:  0.847877358490566
[[354 59]
 [ 70 365]]

```

### Bagging Model - Random Forest

```

[22]: # RF on Count Vectors
accuracy, confusion = train_model(ensemble.RandomForestClassifier(),
    ↪xtrain_count, train_y, xtest_count)
print ("RF, Count Vectors Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# RF on Word Level TF IDF Vectors
accuracy, confusion = train_model(ensemble.RandomForestClassifier(),
    ↪xtrain_tfidf, train_y, xtest_tfidf)
print ("RF, WordLevel TF-IDF Vectors: ", accuracy)

```

```

with tf.Session() as sess:
    print(confusion.eval())

# RF on Ngram Level TF IDF Vectors
accuracy, confusion = train_model(ensemble.RandomForestClassifier(),
    ↪xtrain_tfidf_ngram, train_y, xtest_tfidf_ngram)
print ("RF, Ngram Level TF IDF Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# RF on Character Level TF IDF Vectors
accuracy, confusion = train_model(ensemble.RandomForestClassifier(),
    ↪xtrain_tfidf_ngram_chars, train_y, xtest_tfidf_ngram_chars)
print ("RF, Character Level TF IDFs Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

```

RF, Count Vectors Vectors: 0.6898584905660378

```
[[327 166]
 [ 97 258]]
```

RF, WordLevel TF-IDF Vectors: 0.7004716981132075

```
[[330 160]
 [ 94 264]]
```

RF, Ngram Level TF IDF Vectors: 0.8561320754716981

```
[[378 76]
 [ 46 348]]
```

RF, Character Level TF IDFs Vectors: 0.8042452830188679

```
[[360 102]
 [ 64 322]]
```

### Boosting Model - Xtereme Gradient Boosting

[23]:

```

# Extereme Gradient Boosting on Count Vectors
accuracy, confusion = train_model(xgboost.XGBClassifier(), xtrain_count.tocsc(),
    ↪train_y, xtest_count.tocsc())
print ("Xgb, Count Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Extereme Gradient Boosting on Word Level TF IDF Vectors
accuracy, confusion = train_model(xgboost.XGBClassifier(), xtrain_tfidf.tocsc(),
    ↪train_y, xtest_tfidf.tocsc())
print ("Xgb, WordLevel TF-IDF: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Extereme Gradient Boosting on Ngram Level TF IDF Vectors

```

```

accuracy, confusion = train_model(xgboost.XGBClassifier(), xtrain_tfidf_ngram.
    ↳ tocsa(), train_y, xtest_tfidf_ngram.tocsa())
print ("Xgb, Ngram Level Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Extreme Gradient Boosting on Character Level TF IDF Vectors
accuracy, confusion = train_model(xgboost.XGBClassifier(),
    ↳ xtrain_tfidf_ngram_chars.tocsa(), train_y, xtest_tfidf_ngram_chars.tocsa())
print ("Xgb, CharLevel Vectors: ", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

```

```

Xgb, Count Vectors:  0.7358490566037735
[[318 118]
 [106 306]]
Xgb, WordLevel TF-IDF:  0.7476415094339622
[[311 101]
 [113 323]]
Xgb, Ngram Level Vectors:  0.9221698113207547
[[373  15]
 [ 51 409]]
Xgb, CharLevel Vectors:  0.8867924528301887
[[363  35]
 [ 61 389]]

```

### Redes neurais rasa - Mullti Layer Perceptron

```

[24]: # Função para criar a arquitetura da rede neural
def create_model_architecture(input_size):
    model = Sequential()

    # create hidden layer
    model.add(layers.Dense(100,
                           input_shape=(input_size, ),
                           activation="sigmoid"))

    # create output layer
    model.add(layers.Dense(1, activation='sigmoid'))

    opt = optimizers.Adam()
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['acc'])

    return model

num_classes = 1

# Shallow Neural Network on Count Vectors Vectors

```

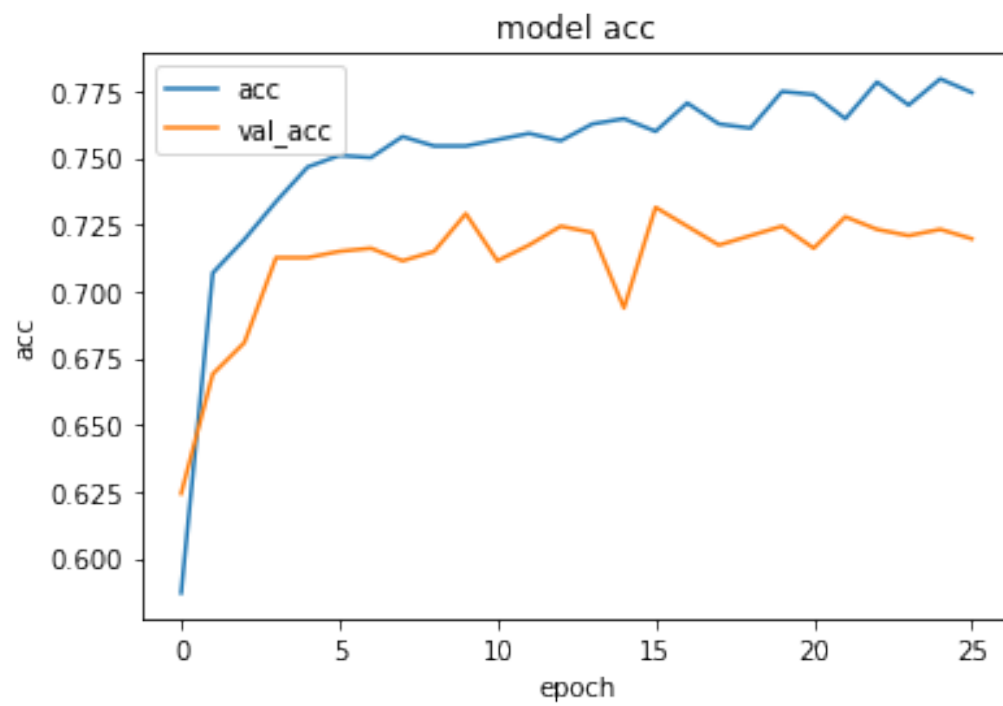
```
classifier = create_model_architecture(xtrain_count.shape[1])
accuracy, confusion = train_model(classifier, xtrain_count, train_y,
    ↪xtest_count, is_neural_net=True)
print ("Shallow Neural Network on Count Vectors Vectors", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Shallow Neural Network on Word Level TF IDF Vectors
classifier = create_model_architecture(xtrain_tfidf.shape[1])
accuracy, confusion = train_model(classifier, xtrain_tfidf, train_y,
    ↪xtest_tfidf, is_neural_net=True)
print ("Shallow Neural Network on Word Level TF IDF Vectors", accuracy,)
with tf.Session() as sess:
    print(confusion.eval())

# Shallow Neural Network on Ngram Level TF IDF Vectors
classifier = create_model_architecture(xtrain_tfidf_ngram.shape[1])
accuracy, confusion = train_model(classifier, xtrain_tfidf_ngram, train_y,
    ↪xtest_tfidf_ngram, is_neural_net=True)
print ("Shallow Neural Network on Ngram Level TF IDF Vectors", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

# Shallow Neural Network on Character Level TF IDF Vectors
classifier = create_model_architecture(xtrain_tfidf_ngram_chars.shape[1])
accuracy, confusion = train_model(classifier, xtrain_tfidf_ngram_chars, train_y,
    ↪xtest_tfidf_ngram_chars, is_neural_net=True)
print ("Shallow Neural Network on Character Level TF IDF Vectors", accuracy)
with tf.Session() as sess:
    print(confusion.eval())
```

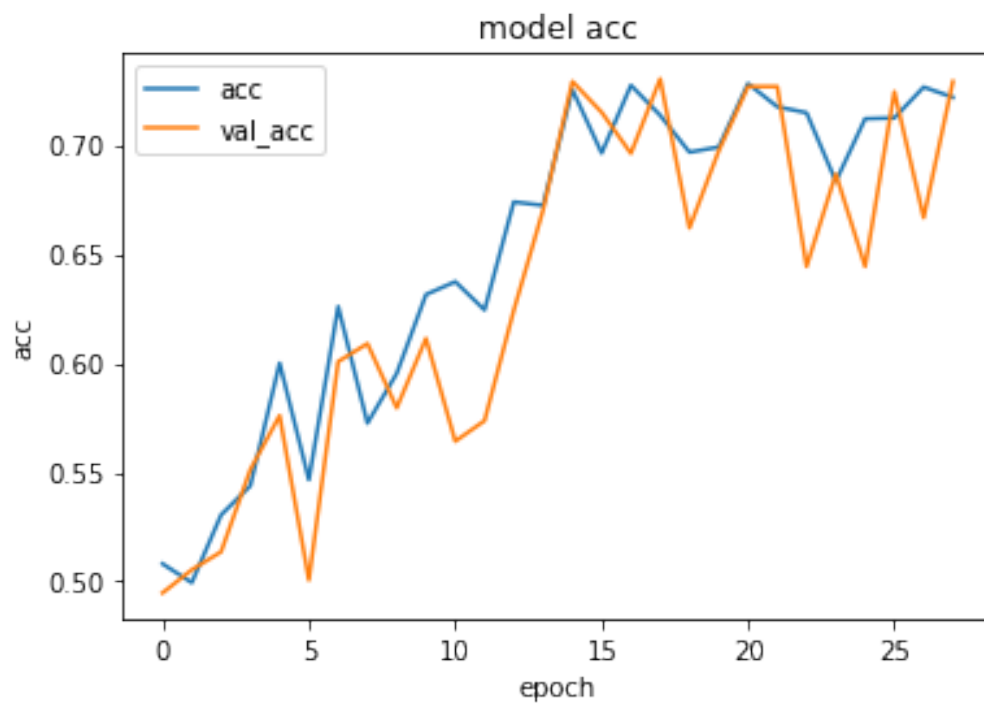




Shallow Neural Network on Count Vectors Vectors 0.7665094339622641

[[327 101]

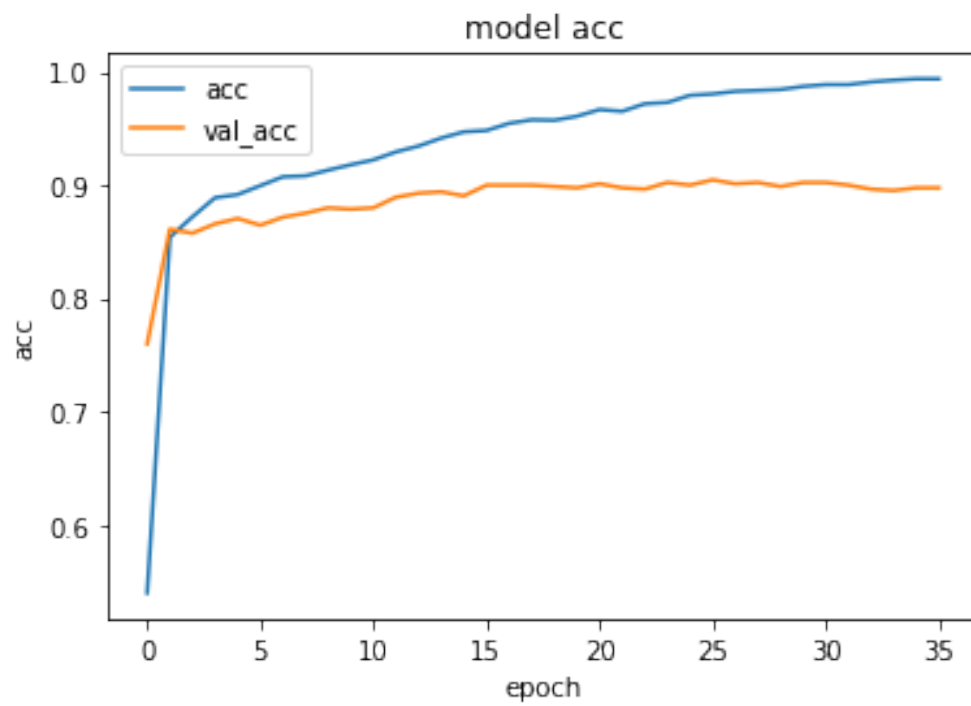
[ 97 323]]



Shallow Neural Network on Word Level TF IDF Vectors 0.7346698113207547

[[295 96]

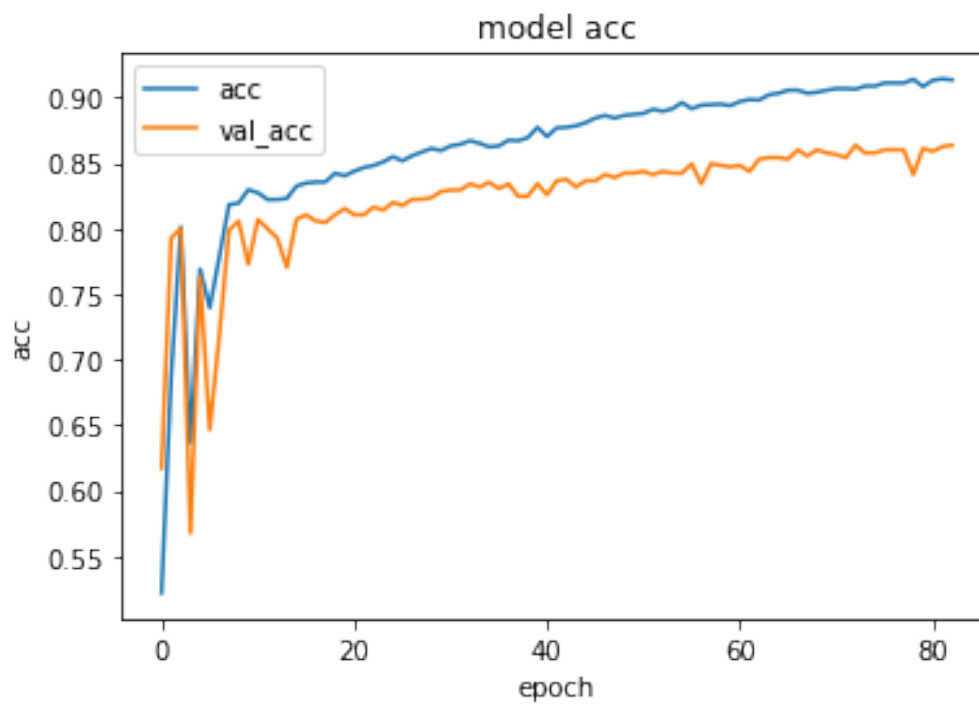
[129 328]]



Shallow Neural Network on Ngram Level TF IDF Vectors 0.9186320754716981

[[390 35]

[ 34 389]]



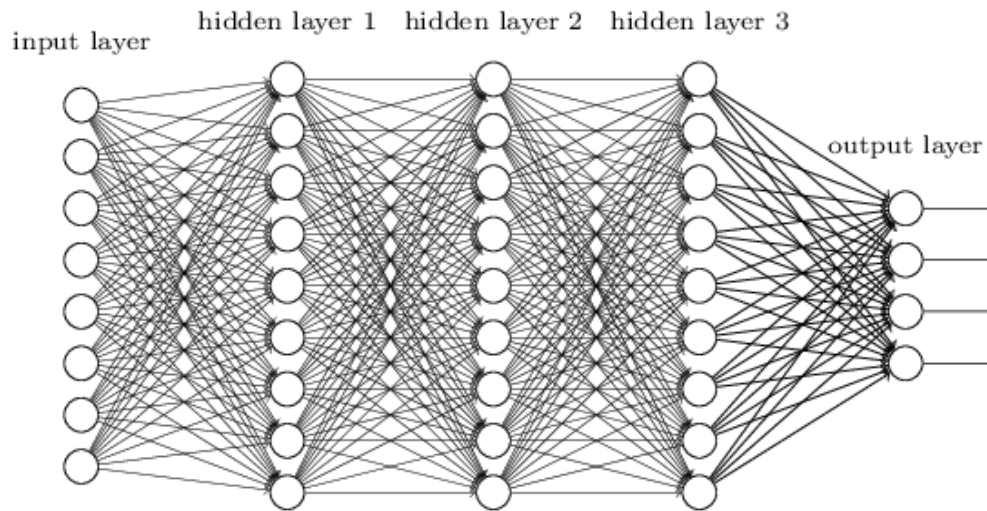
Shallow Neural Network on Character Level TF IDF Vectors 0.8832547169811321

```
[[365 40]
```

```
[ 59 384]]
```

### Deep Neural Networks

Redes neurais profundas são redes mais complexas em que as camadas escondidas realizam operações mais complexas. Diferentes tipos de redes podem ser aplicados aos problemas de classificação de texto.



### Convolutional Neural Network

```
[25]: def create_cnn():
    # Add an Input Layer
    input_layer = layers.Input((maxlen, ))

    # Add the word embedding Layer
    embedding_layer = BDTD_word2vec_50.wv.
    ↪get_keras_embedding(train_embeddings=False)(input_layer)
    embedding_layer = layers.SpatialDropout1D(0.05)(embedding_layer)

    # Add the convolutional Layer e pooling layer
    conv_layer_1 = layers.Convolution1D(128, 5, ↪
    ↪activation="relu")(embedding_layer)
    pooling_layer_1 = layers.MaxPooling1D(2)(conv_layer_1)
    pooling_layer_1 = layers.Dropout(0.15)(pooling_layer_1)

    conv_layer_2 = layers.Convolution1D(128, 5, ↪
    ↪activation="relu")(pooling_layer_1)
    pooling_layer_2 = layers.MaxPooling1D(2)(conv_layer_2)
    pooling_layer_2 = layers.Dropout(0.15)(pooling_layer_2)

    conv_layer_3 = layers.Convolution1D(128, 5, ↪
    ↪activation="relu")(pooling_layer_2)
    pooling_layer_3 = layers.GlobalMaxPooling1D()(conv_layer_3)
    pooling_layer_3 = layers.Dropout(0.15)(pooling_layer_3)

    # Add the output Layers
    output_layer1 = layers.Dense(512, activation="relu")(pooling_layer_3)
    output_layer2 = layers.Dense(512, activation="relu")(output_layer1)
    output_layer2 = layers.Dropout(0.12)(output_layer2)
```

```

output_layer3 = layers.Dense(1, activation="sigmoid")(output_layer2)

# Compile the model
model = models.Model(inputs=input_layer, outputs=output_layer3)
model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy',
metrics=['acc'])
print(model.summary())
return model

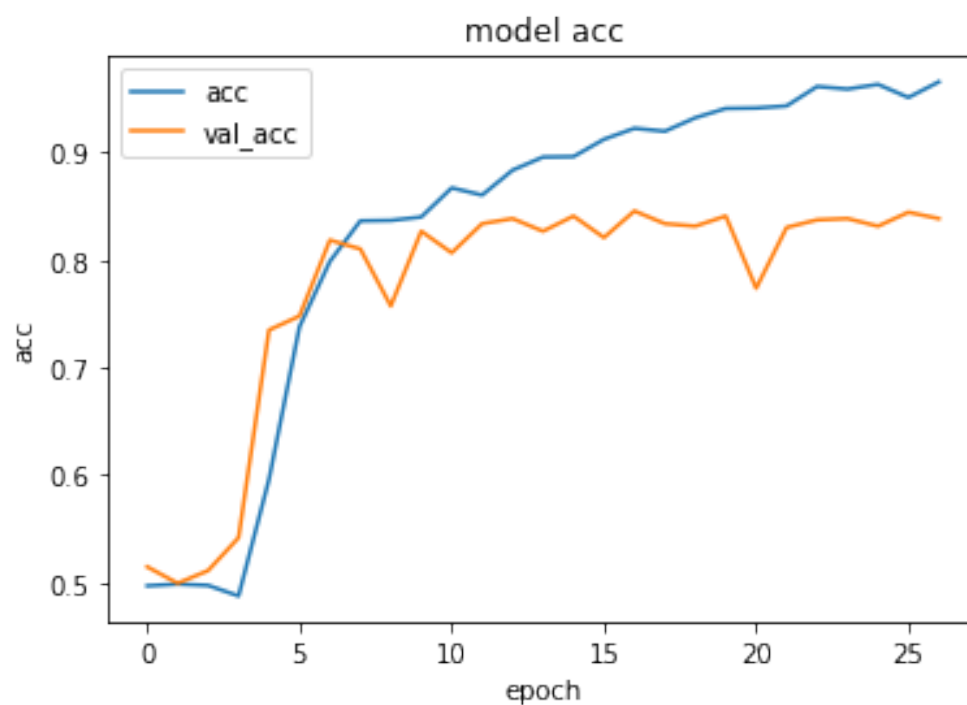
classifier = create_cnn()
accuracy, confusion = train_model(classifier, train_seq_x, train_y, test_seq_x,
is_neural_net=True)
classifier.save("model_cnn.h5")

print ("CNN, Word Embeddings", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 400)	0
embedding_1 (Embedding)	(None, 400, 50)	9289150
spatial_dropout1d_1 (Spatial	(None, 400, 50)	0
conv1d_1 (Conv1D)	(None, 396, 128)	32128
max_pooling1d_1 (MaxPooling1	(None, 198, 128)	0
dropout_1 (Dropout)	(None, 198, 128)	0
conv1d_2 (Conv1D)	(None, 194, 128)	82048
max_pooling1d_2 (MaxPooling1	(None, 97, 128)	0
dropout_2 (Dropout)	(None, 97, 128)	0
conv1d_3 (Conv1D)	(None, 93, 128)	82048
global_max_pooling1d_1 (Glob	(None, 128)	0
dropout_3 (Dropout)	(None, 128)	0

dense_9 (Dense)	(None, 512)	66048
dense_10 (Dense)	(None, 512)	262656
dropout_4 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 1)	513
=====		
Total params: 9,814,591		
Trainable params: 525,441		
Non-trainable params: 9,289,150		
-----		



CNN, Word Embeddings 0.8620283018867925  
 [[336 29]  
 [ 88 395]]

### Recurrent Neural Network – LSTM

```
[26]: def create_rnn_lstm():

    # Add an Input Layer
    input_layer = layers.Input((maxlen, ))
```

```

# Add the word embedding Layer

embedding_layer = BDTD_word2vec_50.wv.
↪get_keras_embedding(train_embeddings=True)(input_layer)

embedding_layer = layers.SpatialDropout1D(0.05)(embedding_layer)

# Add the LSTM Layer
lstm_layer_1 = layers.LSTM(256, return_sequences=True)(embedding_layer)
↪#return_sequences=True
lstm_layer_2 = layers.LSTM(128, return_sequences=True)(lstm_layer_1)
lstm_layer_3 = layers.LSTM(64)(lstm_layer_2)

# Add the output Layers
output_layer1 = layers.Dense(256, activation="relu")(lstm_layer_3)
output_layer1 = layers.Dropout(0.2)(output_layer1)
output_layer2 = layers.Dense(1, activation="sigmoid")(output_layer1)

# Compile the model
model = models.Model(inputs=input_layer, outputs=output_layer2)
model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy',
↪metrics=['acc'])
print(model.summary())
return model

classifier = create_rnn_lstm()
accuracy, confusion = train_model(classifier, train_seq_x, train_y, test_seq_x,
↪is_neural_net=True)
print("RNN-LSTM, Word Embeddings", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 400)	0
embedding_2 (Embedding)	(None, 400, 50)	9289150
spatial_dropout1d_2 (Spatial	(None, 400, 50)	0
lstm_1 (LSTM)	(None, 400, 256)	314368
lstm_2 (LSTM)	(None, 400, 128)	197120
lstm_3 (LSTM)	(None, 64)	49408



dense_12 (Dense)	(None, 256)	16640
------------------	-------------	-------

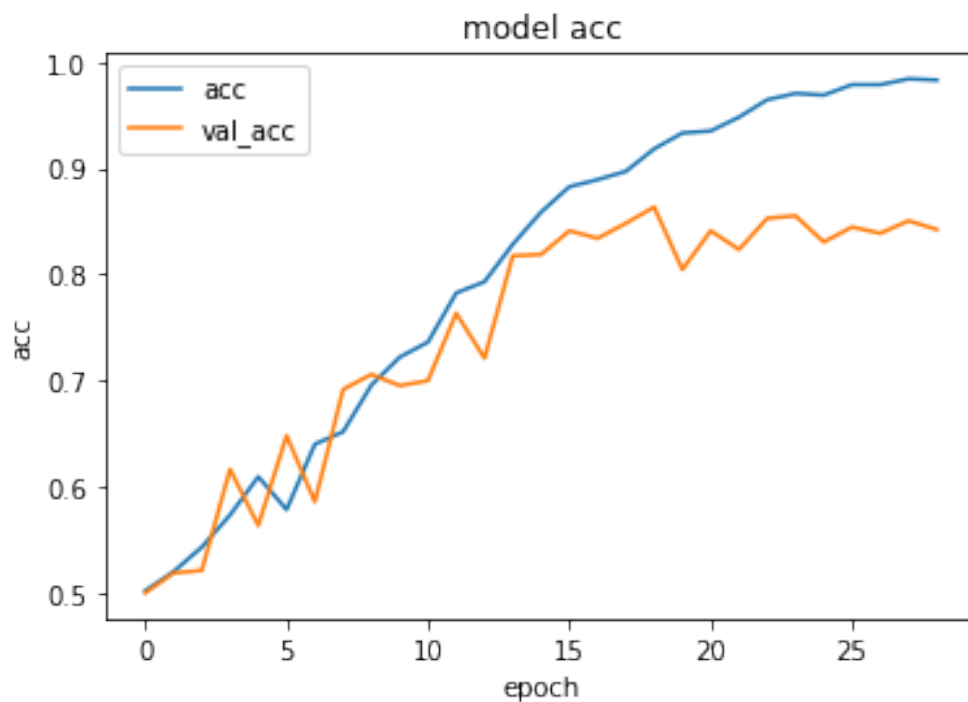
dropout_5 (Dropout)	(None, 256)	0
---------------------	-------------	---

dense_13 (Dense)	(None, 1)	257
------------------	-----------	-----

=====  
Total params: 9,866,943

Trainable params: 9,866,943

Non-trainable params: 0  
=====



RNN-LSTM, Word Embeddings 0.8820754716981132

[[359 35]

[ 65 389]]

### Recurrent Neural Network – GRU

```
[27]: def create_rnn_gru():
    # Add an Input Layer
    input_layer = layers.Input((maxlen, ))

    # Add the word embedding Layer
    embedding_layer = BDTD_word2vec_50.wv.
    ↪ get_keras_embedding(train_embeddings=True)(input_layer)
```

```

embedding_layer = layers.SpatialDropout1D(0.05)(embedding_layer)

# Add the GRU Layer
gru_layer = layers.GRU(20, return_sequences=True)(embedding_layer)
gru_layer_1 = layers.GRU(20)(gru_layer)

# Add the output Layers
output_layer1 = layers.Dense(50, activation="sigmoid")(gru_layer_1)
output_layer1 = layers.Dropout(0.2)(output_layer1)
output_layer2 = layers.Dense(1, activation="sigmoid")(output_layer1)

# Compile the model
model = models.Model(inputs=input_layer, outputs=output_layer2)
model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy',
metrics=['acc'])
print(model.summary())
return model

classifier = create_rnn_gru()
accuracy, confusion = train_model(classifier, train_seq_x, train_y, test_seq_x,
is_neural_net=True)
print ("RNN-GRU, Word Embeddings", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

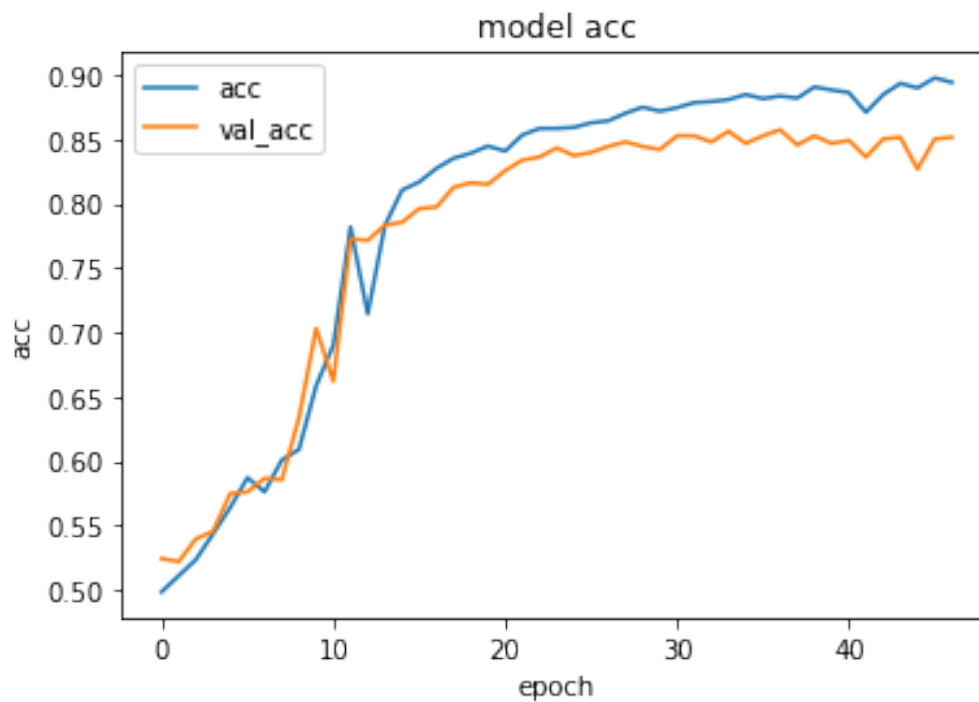
```

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, 400)	0
embedding_3 (Embedding)	(None, 400, 50)	9289150
spatial_dropout1d_3 (Spatial	(None, 400, 50)	0
gru_1 (GRU)	(None, 400, 20)	4260
gru_2 (GRU)	(None, 20)	2460
dense_14 (Dense)	(None, 50)	1050
dropout_6 (Dropout)	(None, 50)	0
dense_15 (Dense)	(None, 1)	51

Total params: 9,296,971  
 Trainable params: 9,296,971

Non-trainable params: 0

---



RNN-GRU, Word Embeddings 0.8726415094339622

[[335 19]

[ 89 405]]

### 3.7.4 Bidirectional RNN

RNN layers can be wrapped in Bidirectional layers as well. Lets wrap our GRU layer in bidirectional layer.

```
[28]: def create_bidirectional_rnn():
    # Add an Input Layer
    input_layer = layers.Input((maxlen, ))

    # Add the word embedding Layer
    embedding_layer = BDTD_word2vec_50.wv.
    get_keras_embedding(train_embeddings=True)(input_layer)

    embedding_layer = layers.SpatialDropout1D(0.05)(embedding_layer)

    # Add the GRU Layer
    lstm_layer = layers.Bidirectional(layers.GRU(20))(embedding_layer)
```

```

# Add the output Layers
output_layer1 = layers.Dense(50, activation="relu")(lstm_layer)
output_layer1 = layers.Dropout(0.2)(output_layer1)
output_layer2 = layers.Dense(1, activation="sigmoid")(output_layer1)

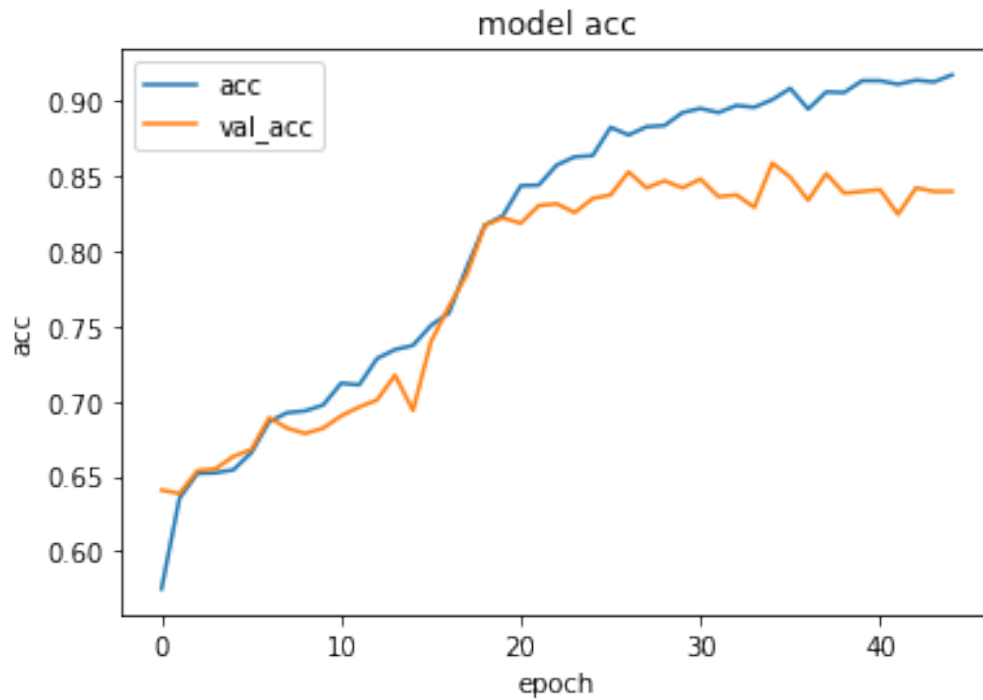
# Compile the model
model = models.Model(inputs=input_layer, outputs=output_layer2)
model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy',
metrics=['acc'])
print(model.summary())
return model

classifier = create_bidirectional_rnn()
accuracy, confusion = train_model(classifier, train_seq_x, train_y, test_seq_x,
is_neural_net=True)
print ("RNN-Bidirectional, Word Embeddings", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

```

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 400)	0
embedding_4 (Embedding)	(None, 400, 50)	9289150
spatial_dropout1d_4 (Spatial	(None, 400, 50)	0
bidirectional_1 (Bidirection	(None, 40)	8520
dense_16 (Dense)	(None, 50)	2050
dropout_7 (Dropout)	(None, 50)	0
dense_17 (Dense)	(None, 1)	51

Total params: 9,299,771  
 Trainable params: 9,299,771  
 Non-trainable params: 0



RNN-Bidirectional, Word Embeddings 0.8620283018867925

[[342 35]

[ 82 389]]

### 3.7.5 Recurrent Convolutional Neural Network

```
[29]: def create_rcnn():
    # Add an Input Layer
    input_layer = layers.Input((maxlen, ))

    # Add the word embedding Layer
    embedding_layer = BDTD_word2vec_50.wv.
    ↪get_keras_embedding(train_embeddings=True)(input_layer)

    embedding_layer = layers.SpatialDropout1D(0.05)(embedding_layer)

    # Add the recurrent layer
    rnn_layer = layers.Bidirectional(layers.GRU(50, ↪
    ↪return_sequences=True))(embedding_layer)

    # Add the convolutional Layer
    conv_layer = layers.Convolution1D(100, 3, ↪
    ↪activation="sigmoid")(embedding_layer)
```

```

# Add the pooling Layer
pooling_layer = layers.GlobalMaxPool1D()(conv_layer)

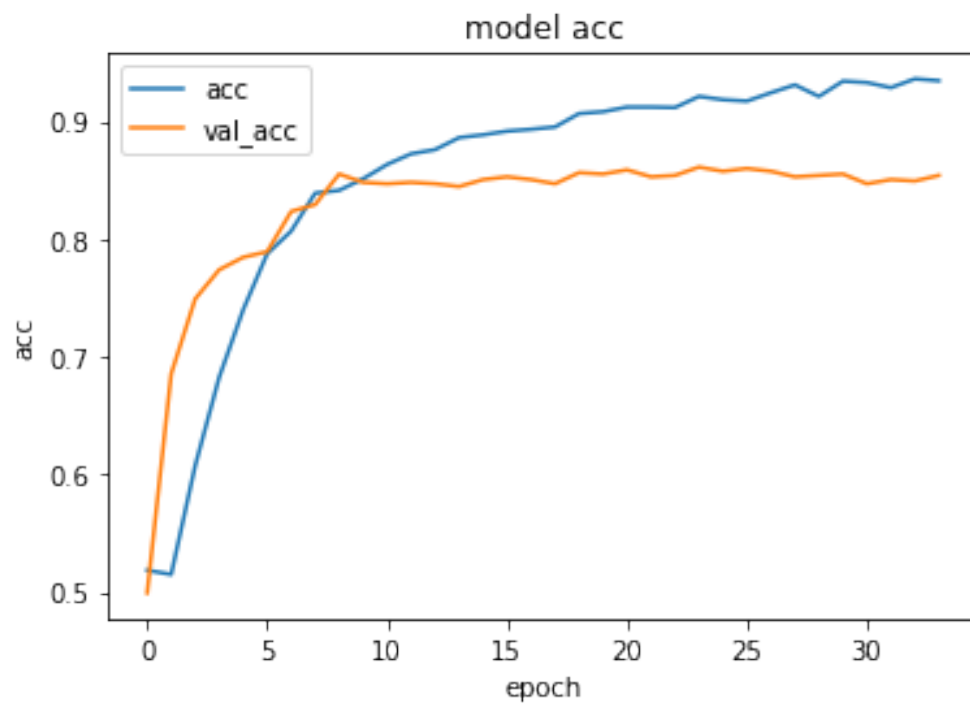
# Add the output Layers
output_layer1 = layers.Dense(50, activation="sigmoid")(pooling_layer)
output_layer1 = layers.Dropout(0.2)(output_layer1)
output_layer2 = layers.Dense(1, activation="sigmoid")(output_layer1)

# Compile the model
model = models.Model(inputs=input_layer, outputs=output_layer2)
model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy',
metrics=['acc'])
print(model.summary())
return model

classifier = create_rcnn()
accuracy, confusion = train_model(classifier, train_seq_x, train_y, test_seq_x,
is_neural_net=True)
print("RCNN, Word Embeddings", accuracy)
with tf.Session() as sess:
    print(confusion.eval())

```

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	(None, 400)	0
embedding_5 (Embedding)	(None, 400, 50)	9289150
spatial_dropout1d_5 (Spatial	(None, 400, 50)	0
conv1d_4 (Conv1D)	(None, 398, 100)	15100
global_max_pooling1d_2 (Glob	(None, 100)	0
dense_18 (Dense)	(None, 50)	5050
dropout_8 (Dropout)	(None, 50)	0
dense_19 (Dense)	(None, 1)	51
Total params: 9,309,351		
Trainable params: 9,309,351		
Non-trainable params: 0		



```
RCNN, Word Embeddings 0.8679245283018868  
[[362 50]  
 [ 62 374]]
```

## 5 Conclusão

O modelo com o melhor resultado encontrado foi o algoritmo Support Vector Machine usando vetores TF-IDF com Bigramas e Trigramas.

## Apêndice 3 - Notebook webscraping repositórios institucionais

Script para buscar resumos na BDTD, testar se eles são relevantes para o domínio de óleo e gás e baixar o documento original no repositório institucional.

```
[1]: import requests
from bs4 import BeautifulSoup as bs
import pandas as pd
import numpy as np
import json
import nltk
from nltk.tokenize import word_tokenize
from langdetect import detect
from langdetect import detect_langs
from keras.models import load_model
import gensim
from gensim.models import Word2Vec
import csv
import re
```

Using TensorFlow backend.

```
[2]: # Definindo configurações globais de proxy para realizar a extração dentro da
      ↪ rede Petrobras
chave = 'XXXX'
pwd = 'XXXXXXXXXX'
proxy_url = 'http://' + chave + ':' + pwd + '@inet-sys.gnet.petrobras.com.br:804/'
proxies = {
    'http' : proxy_url ,
    'https' : proxy_url ,
}
```

Inicialmente entraremos no site da BDTD e buscaremos os links de todas as teses de uma determinada instituição.

```
[3]: #função para coletar os links das tese

def get_links(page):

    #preparar a url
```



```

url = ('http://bdtd.ibict.br/vufind/Search/Results?
↪filter%5B%5D=institution%3A%22UFBA%22&type=AllFields&page=' +
      str(page))

#Fazer requisição e parsear o arquivo html
f = requests.get(url, proxies = proxies).text
soup = bs(f, "html.parser")

#Coletando link para as teses
links = []
for doc in soup.find_all('a', href=True):
    if 'title' in doc.get('class', []):
        links.append(doc['href'])
return links

```

```

[4]: #Coletar o link de todas as teses
start_page = 1
n_pages = 500 # Cada página retorna 20 teses

links = []

for p in range(start_page, n_pages):
    link = get_links(p)
    if link != []:
        links = links + link
    else:
        break

    if p % 100 == 0:
        print (p*20, ' links capturados, ', p, ' páginas')
        with open('links_ufba', "w") as output:
            writer = csv.writer(output, lineterminator='\n')
            for val in links:
                writer.writerow([val])

with open('links_ufba', "w") as output:
    writer = csv.writer(output, lineterminator='\n')
    for val in links:
        writer.writerow([val])
print (p*20, ' links capturados, ', p, ' páginas')

```

```

2000 links capturados, 100 páginas
4000 links capturados, 200 páginas
6000 links capturados, 300 páginas
8000 links capturados, 400 páginas
9940 links capturados, 497 páginas

```

[5]: *# Abrindo arquivo gravado anteriormente*

```
#links = []
#with open('links_ufba', 'r') as f:
#    reader = csv.reader(f)
#    for link in reader:
#        links.append(link[0])
```

Em seguida vamos recuperar os metadados de cada link coletado anteriormente.

[6]: *#função para buscar os metadados das teses no BDTD*

```
def tese_link(link):
    #definir url
    url = 'http://bdtb.ibict.br' + link

    #Requisitar html e fazer o parser
    f = requests.get(url, proxies = proxies).text
    soup = bs(f, "html.parser")

    #Dicionário para armazenar as informações da tese
    tese = {}

    #Adicionar título
    tese['Title'] = soup.find('h3').get_text()
    for doc in soup.find_all('tr'):
        #Identificar atributo
        try:
            atributo = doc.find('th').get_text()
        except:
            pass
        #Verificar se o atributo possui mais de um dado
        for row in doc.find_all('td'):
            #Adicionar o atributo no dicionário
            if row.find('div') == None:
                try:
                    tese[atributo] = doc.find('td').get_text()
                except:
                    pass
            else:
                element = []
                #No dicionário, adicionar todos os dados ao seu respectivo
                ↪atributo
                for e in doc.find_all('div'):
                    try:
                        sub_e = []
                        for sub_element in e.find_all('a'):
                            element.append(sub_element.get_text())
                        #element.append(sub_e)
```

```

        except:
            pass
        tese[atributo] = element

    return(tese)

```

Como em alguns casos o resumo português e inglês se misturaram, foi implementado uma função para separar os textos misturados

[7]: *# Função para separar resumos português e inglês*

```

def separacao_port_engl(abstract):

    mix_sent = nltk.sent_tokenize(abstract)

    new_mix = []
    for sent in mix_sent:
        position = sent.find('.')
        if position != len(sent)-1:
            sent_1 = sent[:position+1]
            sent_2 = sent[position+1:]
            new_mix.append(sent_1)
            new_mix.append(sent_2)
        else:
            new_mix.append(sent)

    mix_sent = new_mix

    port = []
    engl = []

    for sent in mix_sent:
        try:
            if detect (sent) == 'pt':
                port.append(sent)
            else:
                engl.append (sent)
        except:
            pass

    port = " ".join(port)
    engl = " ".join(engl)

    return(port, engl)

```

Até esse momento estamos recuperando as informações de todas as teses de uma determinada instituição. No entanto o objetivo é gravar os metadados e salvar o arquivo apenas das teses relacionadas a O&G. Portanto, vamos carregar os algoritmos de classificação e de vetorização de palavras treinados previamente.

```
[8]: # Carregando modelo Word2Vec
BDTD_word2vec_50 = Word2Vec.load("../...\\Embeddings\\BDTD_word2vec_50")
# Carregando modelo keras
model_keras = load_model('../...\\model_cnn.h5')
model_keras.summary()
```

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	(None, 400)	0
embedding_6 (Embedding)	(None, 400, 50)	9289150
spatial_dropout1d_6 (Spatial	(None, 400, 50)	0
conv1d_13 (Conv1D)	(None, 396, 128)	32128
max_pooling1d_9 (MaxPooling1	(None, 198, 128)	0
dropout_18 (Dropout)	(None, 198, 128)	0
conv1d_14 (Conv1D)	(None, 194, 128)	82048
max_pooling1d_10 (MaxPooling	(None, 97, 128)	0
dropout_19 (Dropout)	(None, 97, 128)	0
conv1d_15 (Conv1D)	(None, 93, 128)	82048
global_max_pooling1d_5 (Glob	(None, 128)	0
dropout_20 (Dropout)	(None, 128)	0
dense_30 (Dense)	(None, 512)	66048
dense_31 (Dense)	(None, 512)	262656
dropout_21 (Dropout)	(None, 512)	0
dense_32 (Dense)	(None, 1)	513
Total params: 9,814,591		
Trainable params: 525,441		
Non-trainable params: 9,289,150		

```
[9]: # dicionário proveniente do modelo de word embedding para converter palavras em
      ↪ índices
word2index = {}
for index, word in enumerate(BDTD_word2vec_50.wv.index2word):
    word2index[word] = index

# Função para converter texto em sequência de índices
def index_pad_text(text, maxlen, word2index):
    maxlen = 400
    new_text = []

    for word in word_tokenize(text):
        try:
            new_text.append(word2index[word])
        except:
            pass
    # Add the padding for each sentence. Here I am padding with 0
    if len(new_text) > maxlen:
        new_text = new_text[:400]
    else:
        new_text += [0] * (maxlen - len(new_text))

    return np.array(new_text)

maxlen = 400
```

Para cada link coletado será feita as seguintes tarefas: \* verificar se o texto português e inglês estão misturados; \* transformar o texto em sequência de índices; \* classificar quanto a relevância ao domínio de O&G; \* se for relevante, gravar os metadados

```
[10]: # Dicionário para agrupar os metadados
metadados = {}
# Contadores de links testados e classificados como O&G
n_test = 0
n_pet = 0
# Testando cada link de links
for link in links:
    n_test += 1
    try:
        # Recuperar o metadados de uma tese
        metadado = tese_link(link)
        # Verificar se existe resumo em inglês, separar texto português/inglês e
        ↪ realocar
        # os textos separados nas respectivas colunas
        if 'Resumo inglês:' not in metadado:
            metadado['Resumo inglês:'] = separacao_port_engl(metadado['Resumo
            ↪ Português:'])[1]
```

```

metadado['Resumo Português:'] = separacao_port_engl(metadado['Resumo_
↳Português:'])[0]
# Colocando o texto em minúscula
text = metadado['Resumo Português:'].lower()
# Convertendo as palavras em sequências de acordo com o modelo word2vec
text_seq = index_pad_text(text, maxlen, word2index)
text_seq = text_seq.reshape((1, 400))
# Usando o algoritmo classificador para prever se a tese é relevante
pred = model_keras.predict(text_seq)[0]
#Se a classificação for menor do que 0.2 manter os metadados
if (pred < 0.2 and len(text) > 100):
    metadado['Classificador'] = pred[0]
    texto_completo = metadado['Download Texto Completo:']
    metadados[texto_completo] = metadado
    n_pet += 1
# Gravando os resultados em JSON
metadados_ufba = pd.DataFrame.from_dict(metadados, orient='index')
metadados_ufba.to_json('metadados_ufba.json', orient = 'index')
print(n_test, " teses avaliadas e ", n_pet, " teses relacionadas a_
↳O&G encontradas.")

except:
    pass

```

9531 teses avaliadas e 187 teses relacionadas a O&G encontradas.

```

[11]: # Incluindo um ID para cada tese
universidade = 'UFBA'
metadados_ufba['PDF_ID'] = metadados_ufba['Download Texto Completo:'].
↳apply(lambda x: universidade +
re.
↳sub('/', '_', x[-6:]))

[12]: metadados_ufba.to_json('metadados_ufba.json', orient = 'index')

[13]: # Carregando arquivos já gravados
metadados_ufba = pd.read_json('metadados_ufba.json', orient = 'index')

```

A próxima etapa será fazer o download das teses classificadas como relevante para o domínio de O&G

```

[14]: for tese in metadados_ufba.iterrows():
    print(tese[1]['PDF_ID'])
    try:
        #preparar a url
        url = tese[1]['Download Texto Completo:']

```

```
#Fazer requisição e parsear o arquivo html
f = requests.get(url, proxies = proxies).text
soup = bs(f, "html.parser")

#Coletando link para arquivo das teses
links = []
for doc in soup.find_all('a', href=True):
    if doc.get_text() == 'View/Open':
        links.append(doc['href'])

#Recuperando e gravando arquivo PDF
url = 'http://repositorio.ufba.br' + links[0]
pdf = requests.get(url, proxies = proxies)
filename = tese[1]['PDF_ID'] + '.pdf'
with open(filename, 'wb') as f:
    f.write(pdf.content)
except:
    pass
```