Cabeçalho%20pagina%20principal

Ronald Alzamende Martins

Previsão do número de depósitos de pedidos de patentes no Instituto Nacional da Propriedade Industrial - INPI utilizando redes neurais

Monografia de Final de Curso

21/04/2021

Monografia apresentada ao Departamento de Engenharia Elétrica da PUC/Rio como parte dos requisitos para a obtenção do título de Especialização em Business Intelligence.

Orientadores:

**Manoela Kohler**

**Felipe Borges**

**Guilherme Strachan**

barra direita

Dedicatória

Dedico este trabalho a minha amada esposa Flávia e filho Felipe. E, em memória do meu irmão caçula Douglas.

Agradecimentos

Agradeço imensamente à paciência e orientação da professora Manoela Kohler, do professor Felipe Borges, do professor Guilherme Strachan, da colega de curso sempre presente Mirley Bitencourt e claro, a toda equipe de professores e monitores do curso BI – Master. Também, devo mencionar a imensa ajuda da técnica em propriedade industrial, a servidora do INPI, Adosinda Maria Lopes pelo auxílio à elucidação de diversas dúvidas a respeito do complexo caminho que um pedido de patente protocolado no INPI segue desde o seu protocolo à decisão final da Autarquia.

**RESUMO**

Esta monografia visa à aplicação das técnicas de localização e uso da informação – LUI, *Data Mining*, *Business Intelligence*, bem como a utilização de ferramentas de previsão de séries temporais por redes neurais em dados contendo informações de pedidos de patentes, publicados na Revista da Propriedade Industrial (RPI) pelo Instituto Nacional da Propriedade Industrial - INPI. Cada depósito de pedido de patente, caso não haja desistência ou arquivamento, é publicado na RPI, contendo informações, de um modo geral, como data do depósito, classificação do pedido de patente (área/campo tecnológico a que pertence o pedido), título, depositante (empresa ou inventor) e país/estado do depositante. Estes dados constituem uma fonte inestimável de informação acerca do panorama tecnológico do país, podendo servir também como fonte de informação para decisões de investimentos em inovação por parte dos principais players de inovação nacionais ou internacionais.

**ABSTRACT**

This work aims to apply the techniques of location and use of information - LUI, Data Mining, Business Intelligence, as well as the use of time series forecasting tools by neural networks in data containing information on patent applications, published in the Industrial Property Magazine (RPI) by the National Institute of Industrial Property - INPI. Each patent application , if there is no withdrawal or filing, is published at RPI, containing information, in general, such as filing date, classification of the patent application (area / technological field to which the application belongs), title, applicant (company or inventor) and country / state of the applicant. These data constitute a valuable source of information about the country's technological panorama, and can also serve as a source of information for investment decisions in innovation by the most relevant national or international innovation players.

Sumário

1. Introdução 6

1.1 MOTIVAÇÃO 9

1.2 OBJETIVOS DO TRABALHO 9

1.3 DESCRIÇÃO DO TRABALHO 10

1.4 ORGANIZAÇÃO DA MONOGRAFIA 10

2. Descrição do Problema 11

3. Metodologia 21

4. RESULTADOS 28

5. Conclusões e TRABALHOS FUTUROS 41

Referências bibliográficas 43

APÊNDICE 44

* 1. Introdução

A inovação é considerada como um dos fatores mais importantes da atualidade para o desenvolvimento de um país ou mesmo à sobrevivência e crescimento de empresas de, praticamente, todos os setores produtivos.

Na literatura especializada é enfatizado que há uma relação direta entre inovação tecnológica, patentes e crescimento tecnológico, e há inúmeras obras ressaltando a importância da proteção da propriedade imaterial no contexto de promoção da inovação (Paranaguá & Reis, 2009) (Macedo & Barbosa, 2000) (Silva & Silva, 2013) (Buainain & Souza, 2018).

O modelo atual de proteção da propriedade intelectual, na maioria dos países democráticos de direito (Brasil, EUA, Japão etc.) é o de possuir um escritório de propriedade industrial, o qual recebe e processa o depósito de pedidos de patentes, o registro de desenho industrial, indicação geográfica, marcas, programa de computador, topografia de circuito integrado etc.

No Brasil, o órgão responsável pela concessão de direitos relativos à propriedade industrial é o INPI – Instituto Nacional da Propriedade Industrial, o qual publica todos os seus atos através da RPI – Revista da Propriedade Industrial (http://revistas.inpi.gov.br/rpi/).

Criado em 1970, o Instituto Nacional da Propriedade Industrial (INPI) é uma autarquia federal vinculada ao Ministério da Economia, que "tem por finalidade principal executar, no âmbito nacional, as normas que regulam a propriedade industrial, tendo em vista a sua função social, econômica, jurídica e técnica, bem como pronunciar-se quanto à conveniência de assinatura, ratificação e denúncia de convenções, tratados, convênios e acordos sobre propriedade industrial", nos termos do art. 2º da Lei nº 5.648, de 11 de dezembro de 1970 (https://www.gov.br/inpi/pt-br/central-de-conteudo/identidade-institucional).

Cada pedido de patente depositado no INPI possui um estado legal, em função de eventos relevantes capazes de alterar a situação do pedido de patente, dentre os quais se destacam: ação externa do usuário (petição), decisão interna do INPI (publicados na RPI) dependendo do tempo decorrido a partir da data de um determinado evento. As decisões ou atos internos do INPI são publicados na RPI com a denominação de “código de despacho” ou códigos INID (Ciancio, Santos, Freeland, Lacerda, & Latsch, 2017). Mais detalhes a respeito de informações de patentes são dados por (Oliveira, Suster, Pinto, Ribeiro, & Silva, 2005).

O INPI divulga, mensalmente e anualmente, estudos quantitativos relativos ao processamento dos pedidos de patentes, marcas etc., fornecendo informações importantes para análise e desenvolvimento de políticas públicas para o crescimento/fortalecimento da inovação tecnológica do País (https://www.gov.br/inpi/pt-br/servicos/patentes/informacao-tecnologica).

A patente de invenção é um monopólio concedido pelo estado, por 20 anos, em troca da informação (divulgação da invenção) relativa à “invenção” per se. Com esta informação, é possível realizar estudos sobre a tecnologia “revelada”, garantindo um fluxo de informação importante para a pesquisa e prospecção de novos produtos. O Escritório Canadense de Propriedade Intelectual (CIPO) delineou perfeitamente algumas principais utilidades de informações disponíveis no banco de dados dos pedidos de patentes (https://www.ic.gc.ca/eic/site/cipointernet-internetopic.nsf/eng/wr03818.html):

* A pesquisa em bancos de dados de pedidos de patentes permite que se aprenda sobre as pesquisas e inovações atuais muito antes de a inovação entrar no mercado;
* No caso do interesse em se desenvolver e comercializar uma inovação específica pode-se pesquisar, nos bancos de dados de patentes e design industrial, para encontrar empresas que já possuem direitos em áreas relacionadas. Podemos encontrar informações sobre direitos que se pode comprar e obter licenças para comercializar ou usar para melhorar inovações já existentes;
* Uma empresa pode acompanhar as patentes no campo tecnológico em que atua, mantendo pesquisas periódicas no banco de dados de patentes, preparando-se para planejar as mudanças, se necessárias, nas condições do mercado. Por exemplo, várias empresas patentearam uma tecnologia que torna possível a cirurgia de longa distância. Uma empresa fornecedora de tecnologia ou serviços de suporte técnico para hospitais poderia utilizar essas informações, encontradas em bancos de dados de patentes, para preparar produtos ou serviços que atendam às novas necessidades dos hospitais que adotam essa tecnologia;
* Posso usar determinada tecnologia na solução de problemas na minha empresa? Pesquisar primeiro no banco de dados de patentes, para ter certeza de que não está pisando na ideia inovadora de outra pessoa, pode evitar dispendiosas dores de cabeça mais tarde. Isto pode ser evitado, utilizando-se dos mecanismos de pesquisa aplicáveis ​​para ver se alguém já tem uma patente ou desenho industrial para sua inovação;
* Pesquisar as tecnologias, no banco de dados de patentes, pertencentes aos mercados de seu interesse pode evitar conflitos desnecessários com outras empresas;
* Evitar possíveis problemas comerciais: se houver muitas patentes ou desenhos industriais para um determinado tipo de inovação semelhante ao seu e ainda não estão disponíveis no mercado, isso pode ser um indicador de que a sua inovação pode não ser comercialmente viável.

Existem duas fontes no Brasil, gratuitas, em que se pode pesquisar sobre os pedidos de patentes depositados no país: o website de busca do INPI (<https://gru.inpi.gov.br/pePI/jsp/patentes/PatenteSearchAvancado.jsp>) e as RPI’s publicadas (http://revistas.inpi.gov.br/rpi/). O portal brasileiro de dados abertos contém recursos para acessar as RPI’s já publicadas pelo INPI (https://dados.gov.br/dataset/revista-da-propriedade-industrial-rpi).

No entanto, uma primeira dificuldade para realizar um estudo, independente das estatísticas publicadas pelo INPI, é justamente o acesso aos depósitos de pedidos de patentes e às informações relevantes contidas nestes depósitos, em um formato passível de análise, isto é, dados de depósito em um período de tempo, no formato de uma tabela/dataframe. Em suma, não há um aplicativo ou software disponibilizado por estes websites do INPI/Governo que forneça as informações no formato de uma tabela/dataframe.

No website do INPI em que são publicadas as RPI’s, desde 05/06/2018, as publicações também possuem um formato xml, o qual, ao ser aberto pelo software excel, por exemplo, já mostra o arquivo organizado/estruturado em uma tabela/dataframe. Porém, em datas anteriores, os arquivos se encontram em formatos txt ou pdf, não sendo possível a visualização do arquivo no formato estruturado. Os dados compilados neste trabalho em formato de uma tabela/dataframe têm como fonte de inspiração a tabela gerada pelo arquivo xml mencionado acima.

Uma primeira possibilidade de trabalho seria a de extrair informações relevantes de cada pedido de patente, diretamente a partir do website de busca do INPI, utilizando-se técnicas de webscraping. No entanto, é sabido que o website de busca, ao se realizar buscas gerais em um período de tempo longo (um mês ou mais), em algum momento da extração, pode retornar um “erro na página”, ocorrendo quedas constantes no website, devido a instabilidades causadas pelo uso simultâneo de vários usuários, podendo vir a ser um trabalho relativamente dispendioso quanto ao tempo ou mesmo do ponto de vista do custo computacional.

Por esta razão, optou-se por extrair estas informações das RPI’s publicadas, pois são dados de acesso livre e público, e uma vez extraídas, estarão armazenadas e disponíveis a qualquer momento, independente da estabilidade do website do INPI. Também, a partir do histórico de despachos publicados na RPI, é possível deduzir o trâmite processual de um pedido de patente protocolado (Ciancio, Santos, Freeland, Lacerda, & Latsch, 2017). Mais detalhes sobre o trâmite processual serão descritos no capítulo 2.

* 1. MOTIVAÇÃO

O fato de não haver um aplicativo de interface (API) para extração de informações disponibilizadas nos websites do INPI em datas anteriores a 05/06/2018 motivou a principal pergunta que norteou esta monografia, a princípio: é possível extrair informações da RPI e transformar estas informações em um formato passível de análise? Isto é, podemos transformar os dados das RPI’s em tabela/dataframe para obter insights sobre as tecnologias com maior quantidade de depósitos no país? Com estes dados será possível desenvolver um modelo de previsão com redes neurais para o número de pedidos publicados na RPI?

* 1. OBJETIVOS DO TRABALHO

Portanto, para responder as perguntas anteriores, o presente trabalho tem como principais objetivos:

* Desenvolver um programa, na linguagem Python, para extração de informações referentes aos depósitos de pedidos de patentes publicados das RPI’s;
* A partir das informações extraídas, criar *gráficos* informativos, no próprio Python, compilando estatísticas sobre: o número de pedidos de patentes depositados publicados no Brasil, as tecnologias com maior número de depósitos no país, os países com maior número de pedidos de patentes, os estados brasileiros que possuem maior número de depósitos, os despachos relevantes emitidos pelo INPI anualmente e;
* Desenvolver um modelo de previsão de depósitos de pedidos publicados na RPI através de redes neurais.
  1. DESCRIÇÃO DO TRABALHO

Esta monografia possui quatro etapas, a saber: Coleta de dados, ETL – Extração, Transformação e Carregamento dos dados, Visualização e Exploração dos dados e modelagem de previsão dos pedidos de patente publicados, por redes neurais.

A coleta de dados será realizada utilizando as bibliotecas selenium e beatifulsoup do Python. As revistas ficarão armazenadas no google drive.

Para a extração dos dados das RPI’s será desenvolvido um programa em Python, o mesmo será feito para a transformação e carregamento dos dados.

Com relação à visualização e exploração dos dados utilizaremos também o Python, com as bibliotecas plotly express, matplotlib e animation mostrando diversos gráficos que podem ser desenvolvidos a partir do dataframe criado, bem como descrever algumas características dos depósitos de pedidos de patentes publicados ao longo dos anos e as principais tecnologias contidas nestes depósitos.

Para o modelo de previsão da quantidade de pedidos de patentes publicados na RPI, serão utilizadas as bibliotecas Keras, Sklearn. O modelo de redes neurais será o LSTM. Serão feitos alguns testes com relação aos otimizadores Adam e SGD e com relação às camadas de neurônios necessárias e também com relação às épocas de treinamento. A métrica de comparação para a escolha do melhor modelo de previsão para os dados será o RMSE (Root Mean Square).

* 1. ORGANIZAÇÃO DA MONOGRAFIA

Esta monografia está dividida em quatro capítulos adicionais, descritos a seguir:

O capítulo 2 descreve detalhadamente o problema a que se propõe resolver com esta monografia, informando quais modelos vem sendo utilizados para a previsão de depósitos de pedidos de patentes em outros escritórios/países bem como identifica as limitações deste trabalho.

O capítulo 3 apresenta, sucintamente, as técnicas empregadas e os programas desenvolvidos.

Já o capítulo 4 apresenta os resultados alcançados e algumas visualizações gráficas a respeito das tecnologias com maior quantidade de pedidos de patentes publicados.

Finalizando, o capítulo 5 descreve as conclusões do trabalho e identificar possíveis trabalhos futuros.

* 1. Descrição do Problema

De antemão, pode-se afirmar que é imperativo descrever, mesmo que suscintamente, como ocorre o processamento de pedidos de patentes protocolados no INPI, para esclarecer o problema que se pretende resolver nesta monografia.

No trabalho “Criação de uma Máquina de estado para os processos de patentes do INPI” (Ciancio, Santos, Freeland, Lacerda, & Latsch, 2017), realizado por um grupo de pesquisadores do INPI, é realizado o mapeamento de todo trâmite processual de um pedido de patente, de forma completa e com a devida acurácia. Com base no estudo citado, tentaremos elucidar o intricado processamento de patentes no INPI.

O ciclo de vida de um pedido de patente se inicia quando o depositante protocola o pedido de patente no INPI. A partir do protocolo inicial, o pedido pode seguir por duas vias: via PCT - Tratado de Cooperação em Matéria de Patentes ou via CUP – Convenção da União de Paris (ou simplesmente pela via de entrada de um pedido com depositante/inventor nacional). É importante elucidar que, apesar da crença do senso comum, não existem patente mundiais, isto é, a patente só tem validade no país em que o pedido foi depositado e concedido o monopólio.

A figura 1 resume como é feito o depósito de patentes via PCT, juntamente com a linha de tempo. A figura 2 compara as linhas de tempo para os pedidos via CUP e PCT.

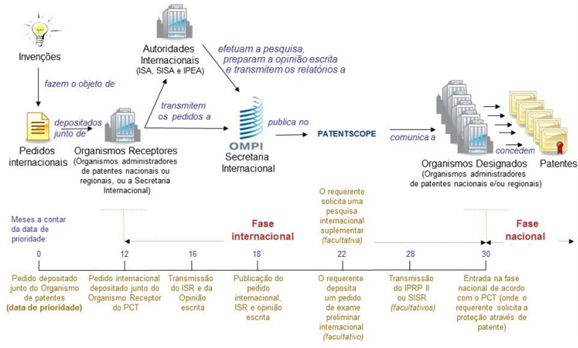


Figura 1 – Resumo do sistema do PCT. Fonte: https://www.wipo.int/export/sites/www/pct/pt/basic\_facts/faqs\_about\_the\_pct.pdf

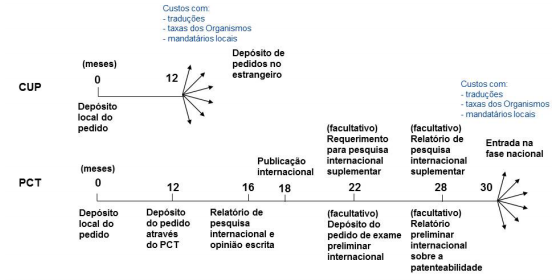


Figura 2 – Comparação entre a via da CUP e a via do PCT. Fonte: WIPO.

Seguindo o fluxo processual no Brasil, os depósitos de pedidos de patentes pela via PCT (https://www.gov.br/pt-br/servicos/solicitar-patente-de-invencao-para-pedido-internacional-pct), após atender a certos requisitos mínimos, terão uma publicação na RPI com despacho 1.1. Se o pedido não foi retirado ou não houve uma desistência ou arquivamento por não atender às requisições necessárias, aí ocorrerá a publicação com despacho 1.3. Juntamente com este despacho publicado na RPI, também são publicados dados relevantes, como: nº do pedido de patente, data do depósito, classificação internacional do pedido de patentes (IPC), título, nome do depositante (pode ser uma empresa ou somente um inventor) e país de origem (no caso do depositante ser nacional também é informado o estado). O depositante tem um prazo de 36 meses, a partir do depósito internacional, para requerer o pedido de exame, sendo arquivado caso não solicite o exame do pedido. Com o requerimento, o pedido segue o fluxo processual, entrando na fila de pedidos de patentes a serem analisadas caso todas as anuidades sejam pagas nos prazos previstos.

Os depósitos de pedidos de patentes nacionais e internacionais que entraram na fase nacional via tratado CUP (https://www.gov.br/inpi/pt-br/backup/legislacao-1/cup.pdf), atendendo a certos requisitos mínimos, terão um código de despacho 2.1 publicado na RPI. Com este código é publicado o número do pedido, a data do depósito e o nome do depositante com país de origem. Estes pedidos serão mantidos em sigilo durante dezoito meses a contar da data de prioridade mais antiga, podendo ser publicado antecipadamente, caso o depositante solicite. Não sendo requerida a publicação antecipada, após o prazo de 18 meses, e caso não haja desistência ou arquivamento e o pedido atenda a uma série de requisitos, aí então será publicado o código 3.1. No caso da publicação antecipada ser requerida e o pedido atender aos requisitos necessários, o código 3.2 será publicado. Tanto o código 3.1 quanto o código 3.2 publicados terão informações relevantes como: nº do pedido de patente, data do depósito, classificação internacional do pedido de patentes (IPC), título, nome do depositante (pode ser uma empresa ou somente um inventor) e país de origem (no caso do depositante ser nacional também é informado o estado). Assim como ocorre com os pedidos de entrada via PCT, o depositante tem um prazo de 36 meses, a partir do depósito, para requerer o pedido de exame, sendo arquivado caso não solicite o exame do pedido. Com o pedido do exame, o pedido segue o fluxo processual, entrando na fila de pedidos de patentes a serem analisadas caso todas as anuidades sejam pagas nos prazos previstos.

Em alguns casos, o pedido original pode ser dividido, por opção do depositante/inventor ou por exigência do próprio INPI. A divisão do pedido significa que existe mais de uma matéria tecnológica a ser patenteada. Assim, a parte do pedido original que foi dividida possui nova numeração, com os mesmos dados bibliográficos do pedido original, com exceção do título e classificação. A publicação do pedido dividido é pelo número de despacho 2.4 ou 23.1.1. Vamos considerar o pedido dividido, pois o mesmo também passa pelo mesmo trâmite processual que o seu original. Vale ressaltar que as publicações 2.4 não contêm informações como a nova classificação ou o título, ficando os dados como desconhecidos no dataframe.

Outro dado, não menos importante, que temos que levar em consideração, é o fato de que os números dos pedidos podem ser alterados, durante o seu processamento, por conta de mudança da natureza (patente de invenção para patente de modelo de utilidade ou vice-versa) ou pelo pedido ter sido publicado com numeração indevida. Para extrair estes dados, a função de extração retornará outro dataframe com o número do pedido e a nova numeração, para posterior ajuste dos dataframes principais. Os despachos que publicam a nova numeração do pedido são 15.10, 15.12 e 51.

Após o exame do pedido, por um pesquisador do INPI, o pedido pode ter as seguintes publicações na RPI: 6.1 (código de despacho para exigência técnica, possui uma forte indicação de futuro deferimento do pedido), 7.1 (código de despacho para ciência, possui uma forte indicação de futuro indeferimento do pedido), 9.1 (código de despacho para deferimento do pedido) e 9.2 (indeferimento do pedido por não atender às exigências técnico-legais). Os despachos 6.20, 6.21 e 6.22 foram despachos criados em 2018/2019, oriundos de um projeto maior que visa à redução do backlog do INPI (https://www.gov.br/inpi/pt-br/servicos/patentes/plano-de-combate-ao-backlog/plano-de-combate-ao-backlog). Caso o depositante não responda aos despachos 6.1, 6.20, 6.21, 6.22 ou 7.1, no prazo de 90 dias, o pedido será arquivado definitivamente, sendo publicado o despacho 11.2.

Existem inúmeros outros despachos relativos ao trâmite processual do pedido, como o 8.6, em que o pedido é arquivado por falta de pagamento da anuidade. O 8.12 trata do arquivamento definitivo. Têm-se os despachos 11.1.1, 11.4, 11.5, 11.6, 11.11 e 11.30 que tratam de arquivamentos por outros motivos. Alguns destes despachos já não são mais utilizados.

Todos os requisitos que os pedidos de patentes devem possuir são as exigências legais regidas pela Lei da Propriedade Industrial – LPI Nº 9.279 de 14 de maio de 1996. Esta lei regula os direitos e obrigações relativos à propriedade industrial no Brasil (http://www.planalto.gov.br/ccivil\_03/leis/l9279.htm). No INPI existem Atos Normativos, Resoluções e diretrizes, todos com base na LPI, que especificam detalhes e pormenores destas exigências legais.

O fluxo processual para pedidos depositado no Brasil pode ser visualizado, com mais detalhes no website https://www.gov.br/inpi/pt-br/servicos/patentes/guia-basico/fluxo-processual-patentes.pdf/view.

Em face do exposto na breve descrição do processamento de patentes no INPI, pode-se delimitar o presente estudo nos seguintes aspectos: os dados extraídos das RPI’s serão os dados referentes aos pedidos de patentes que obtiveram os despachos 1.3, 3.1, 3.2, 2.4 e 23.1.1. Isto significa que não serão trabalhados todos os pedidos de patentes protocolados no INPI, mas somente aqueles que chegaram aos despachos 1.3, 3.1, 3.2, 2.4 e 23.1.1, ou seja, todos os pedidos que atendem a certos requisitos mínimos iniciais para seguirem o fluxo processual, sendo esta quantidade obviamente menor que o número total de pedidos de patentes protocolados na autarquia.

As informações de interesse extraídas dos despachos supracitados serão: o nº do pedido de patente, a data do depósito, o número do pedido original se for pedido dividido, a classificação internacional do pedido de patentes (IPC), o título, o nome do depositante (pode ser uma empresa ou somente um inventor) e o país de origem (no caso do depositante ser nacional também é informado o estado). A IPC indica, basicamente, a qual tecnologia pertence o pedido de patente (https://www.gov.br/inpi/pt-br/servicos/patentes/classificacao-de-patentes)

Também serão extraídos das RPI’s os despachos 6.1, 6.20, 6.21, 6.22, 7.1, 8.6, 8.11, 8.12, 9.1, 9.2, 11.1.1, 11.2, 11.4, 11.5, 11.6, 11.11, 11.18, 11.30 e 11.31 e suas respectivas datas de publicação (data da publicação da respectiva RPI) para termos uma ideia da evolução das primeiras ações e decisões com relação às patentes de invenção depositadas no INPI.

No intuito de simplificar a análise, não serão levados em consideração as publicações anuladas. Com relação aos despachos 1.3, 3.1, 3.2, 2.4 e 23.1.1, caso haja republicações, será levado em consideração somente a última data de publicação de cada despacho, o que na verdade não altera a data de depósito do pedido. Com relação aos outros despachos, serão consideradas todas as publicações, também para efeito de simplificação.

O primeiro problema a ser resolvido, na extração dos dados, é o da coleta das RPI’s. A partir de janeiro de 1993, todas as RPI’s estão disponíveis no formato txt. Antes disto, existem muitas revistas publicadas somente em pdf, um formato mais complexo para extração dos dados. Portanto, o período inicial de extração será de 01/01/1993. O período final será o de 08/12/2020. Ao todo são 1453 RPI’s publicadas no período citado. Muitas RPI’s, senão todas, ao se fazer o download, estão “zipadas”, outro problema a ser resolvido, pois teremos que tirar os arquivos txt da pasta “zipada”, para poder extrair as informações.

O segundo problema é o da extração dos dados propriamente ditos. Nas revistas mais antigas, as publicações não eram separadas, como é atualmente, em que há uma publicação para despachos relativos aos pedidos de patentes, uma para desenhos industriais e assim por diante, ou seja, era tudo publicado em uma revista/arquivo somente. Assim, nos despachos 1.3, 3.1, 3.2, 2.4 e 23.1.1 havia informações a respeito de outros pedidos, como modelo industrial, desenho industrial etc.

Os despachos são dispostos linha a linha, no corpo da RPI, como mostrado na figura 3.

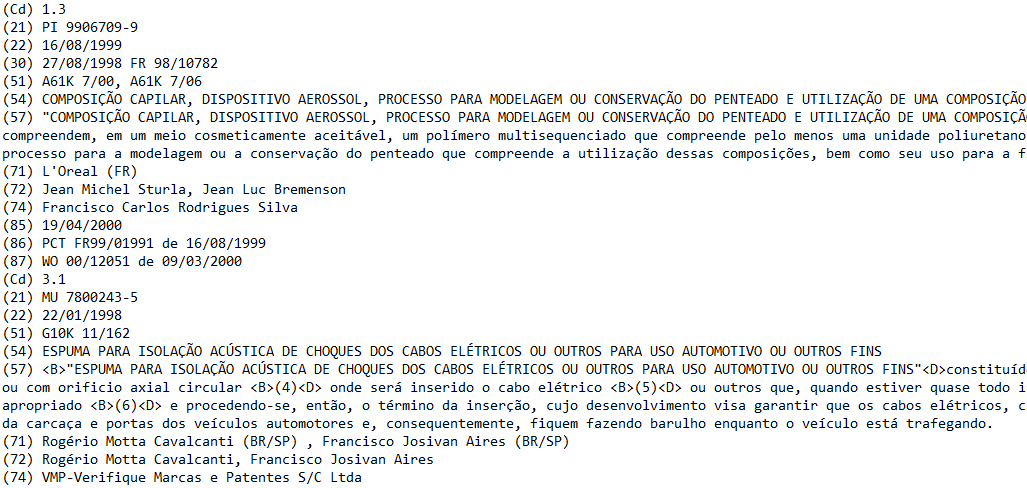


Figura 3 – Visualização das informações contidas nos despachos 1.3 e 3.1

da RPI 1544 de 08/08/2000.

Na figura 3 vemos um formato geral em que aparecem as informações de cada despacho que serão extraídas para este trabalho, cuja ordem geral é:

(Cd) Código de despacho

(21) Número do pedido de patente

(22) data do depósito

(51) classificação IPC

(54) título do pedido de patente

(71) depositante do pedido de patente, pode ser uma empresa ou um inventor mais o país de origem do depositante entre parênteses.

No entanto, não é sempre que a ordem após o despacho apresenta esta sequência, alguns pedidos possuem prioridade unionista cujo código é (30), um dado que não fará parte da análise. Outro dado que as RPI's mais antigas possuem é o resumo (código (57)), que segue logo após a linha do título e, infelizmente, nas RPI's mais recentes o resumo foi excluído e, portanto, não será utilizado para a análise. Além disso, nas RPI's mais recentes, além da ipc, o pedido também apresenta mais uma linha, com o código (52), que é a classificação cpc, outro dado que também não é do nosso interesse. Em alguns pedidos, pode não ter a classificação (ipc), pode não ter título e pode não ter o nome do depositante também. O que dificulta mais o desenvolvimento de um programa para a extração.

Na figura 4 podem ser visualizadas as informações contidas no despacho 6.1. Assim como acontece com os despachos 1.3, 3.1 e 3.2, também podem faltar algumas informações, como data do depósito ou depositante.

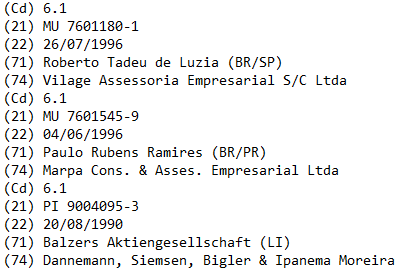


Figura 4 – Visualização das informações contidas no despacho 6.1 da RPI 1544 de 08/0/2000.

Para podemos realizar algumas análises, será necessário transformar os registros das RPI’s em uma tabela/dataframe com o seguinte formato geral:

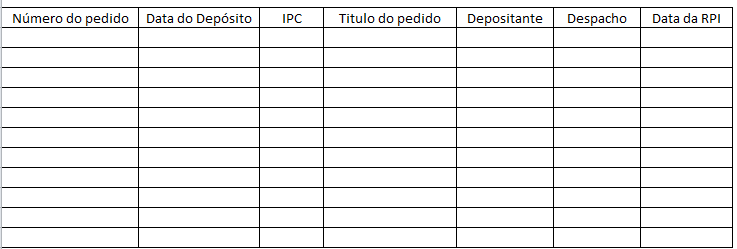


Figura 5 – Tabela a ser formada com as informações extraídas das RPI’s

Com relação à linguagem de programação a ser utilizada, pode-se questionar, por que usar o Python?

A linguagem Python vem se consolidando como uma das principais ferramentas da ciência de dados. Em um estudo sobre os registros de marcas no INPI (Pereira, 2018), o autor utilizou algoritmos em Python para a extração dos dados de interesse da RPI. Foram estudados aspectos de desempenho do INPI com relação às concessões de marcas, utilizando dados de publicações das RPI’s no período de 07/2013 a 06/2019.

Como bem apontado por estudo acima, as estatísticas publicadas pelo INPI podem diferir de estatísticas deduzidas a partir dos despachos publicados nas RPI’s, pois o INPI, por seu acesso irrestrito ao banco de dados, tem dados completos de todos os depósitos/registros protocolados na autarquia. No estudo, foi informado que os algoritmos utilizados em seu trabalho serão parte de um software amplo e robusto para aplicação na área, o que não é objetivo da presente monografia.

A própria cartilha para desenvolvedores publicada pelo portal de dados abertos do governo federal cita o Python como uma das linguagens mais populares para a implementação de scrapers (Cartilha para desenvolvedores, dados abertos. Governo Federal, 2011).

Desta forma, os algoritmos de extração para o presente estudo serão desenvolvidos na linguagem Python.

A importância e utilização de análises tendo como base os dados de propriedade intelectual/industrial também não é novidade.

Em uma revisão sobre o estado da arte em análise de patentes (Abbas, Zhang, & Khan, 2014) são mapeadas ferramentas capazes de realizar uma ampla gama de tarefas, como analisar e prever tendências tecnológicas, realizar o planejamento estratégico de tecnologia, detectar a violação de patentes, determinar a qualidade de patentes e as patentes mais promissoras. Além disso, as principais características e limitações das ferramentas e técnicas utilizadas são apresentadas e vários direcionamentos para pesquisas futuras são destacadas.

No estudo “Procurando por inovação? Que tal usar uma ferramenta gratuita para procurar em 110 milhões de patentes?” (Di Petta & Ferraz, 2020) é utilizada a ferramenta Patant2Net (P2N) para minerar dados de patentes da base Espacenet. Neste estudo, a mineração das informações teve como objetivo extrair informações e identificar novas oportunidades de negócio para pequenos empreendedores através da análise de documentos de patentes publicados, mostrando que é possível definir temas de interesse e identificar tecnologias livres de proteção no Brasil, podendo as mesmas serem aperfeiçoadas ou até mesmo aplicadas diretamente para solução de problemas nas empresas. A figura 6 mostra um fluxograma genérico para análise de patentes.

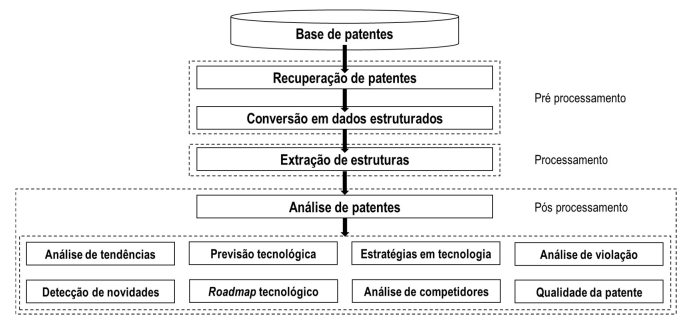


Figura 6 – Fluxo genérico para análise de dados de patentes (Di Petta & Ferraz, 2020), adaptado de (Abbas, Zhang, & Khan, 2014).

Em um estudo sobre a revisão da literatura sobre inteligência artificial, aprendizado de máquina e deep learning para analisar dados de propriedade intelectual (Aristodemou & Tietze, 2018), assim como feito no estudo citado acima, são listadas, em ordem alfabética, as principais técnicas empregadas. Estas técnicas podem ser visualizadas na figura 7.

A seguir, serão citados alguns estudos, que podem ser utilizados na fase de , na fase pós-processamento, os métodos de previsão, tal como mostrado na figura 6. Pois, um dos objetivos deste trabalho é o de desenvolver um modelo de redes neurais para prever a quantidade de pedidos de patentes publicados.

No estudo do uso de métodos de previsão para pedidos de patentes e marcas na Espanha (Hidalgo & Gabaly, 2012) são comparados métodos de previsão para depósitos de patentes e marcas na Espanha. Foram utilizados dois modelos de previsão: modelos de regressão e modelos de séries temporais. Os resultados do estudo, no caso da Espanha, mostraram que é possível utilizar diferentes modelos de previsão, especialmente o ARIMA, com ajuste satisfatório e erro relativamente baixo.

Em “um estudo de previsão de números de patentes por regressão linear em tecnologia de armazenamento em nuvem” (Kuotsan & Chen, 2014) foram utilizados modelos de regressão linear para a previsão de número de patentes.

Já no estudo “um modelo de regressão linear logarítmica dinâmica para prever o número de depósitos de patentes no Escritório de Patentes Europeu” (Hingley & Park, 2015) aplicaram um modelo de regressão log-linear dinâmica para a previsão de depósitos de patentes no Escritório de Patentes Europeu (European Patent Office – EPO). Neste estudo um modelo econométrico é utilizado, levando-se em consideração variáveis como: investimentos em pesquisa e desenvolvimento e Produto Interno Bruto (PIB).

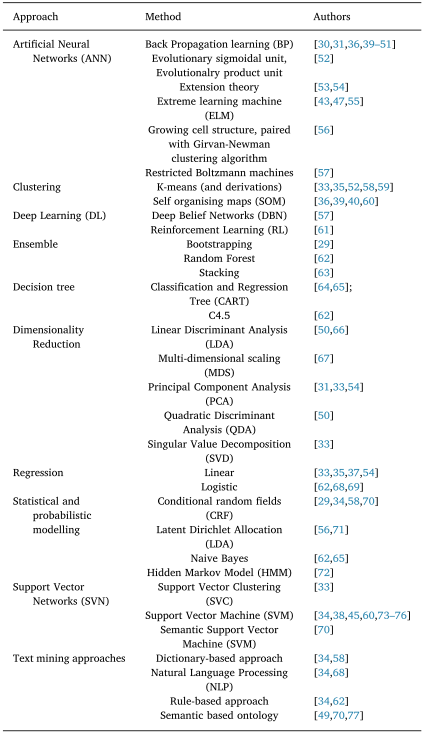


Figura 7 – Métodos de análise aplicados a dados de propriedade intelectual. Os autores citados na tabela podem ser consultados em (Aristodemou & Tietze, 2018).

Em outro estudo (Markovic, Petkovic, Nikolic, & Milovancevic, 2017), foram analisados os dados de patentes depositados em três países (Sérvia, Croácia e Hungria) e a influência de diferentes parâmetros nos depósitos. O método ANFIS (sistema de inferência neuro fuzzy adaptativa) foi aplicado para selecionar os parâmetros mais importantes para o depósito de patentes. Estes parâmetros de entrada são: encargos pelo uso de propriedade intelectual para pagamentos e recebimentos, investimentos em pesquisa e desenvolvimento, registro de marcas para residentes e não residentes, pesquisadores em pesquisa e desenvolvimento (P&D), técnicos em P&D e exportações de alta tecnologia. Como saída tem-se: depósito de patentes para não residentes e residentes. O resultado do estudo mostrou que a combinação de investimentos em P&D e técnicos em P&D são os fatores que tem mais influência para o depósito de patentes. Para avaliar a importância da entrada foi utilizado o indicador RMSE.

No estudo “prevendo o sucesso de novas tecnologias baseado em dados de patentes usando modelos de redes neurais” (Saad, Jneid, & Saleh, 2020), os dados de patentes foram extraídos do Escritório de Marcas e Patentes dos Estados Unidos (USPTO) e usados para prever o crescimento futuro de patentes de duas tecnologias candidatas: "Computação em Nuvem” e “Veículos Autônomos”. Esta abordagem é implementada usando dois modelos de redes neurais para comparação de precisão: uma rede neural ampla e profunda (WDNN) e uma rede neural recorrente (RNN). Como resultado, a RNN atinge um melhor desempenho e precisão e supera o WDNN no contexto de pequenos conjuntos de dados.

* 1. Metodologia

Adaptando o fluxo de trabalho para análise de dados de patentes da figura 6 para o presente estudo (Figura 8), temos:

Figura 8 – Fluxo de trabalho utilizado

Na etapa de pré-processamento, o download das RPI’s será feito de forma automática, por um programa desenvolvido em Python, no ambiente do google colab (https://colab.research.google.com/notebooks/intro.ipynb), o qual permite armazenamento e processamento em nuvem. A conversão em dados estruturados e transformação também será em Python no ambiente do google colab.

Na etapa do pós-processamento, a análise, exploração e representação dos dados será feita também no Python, assim como o modelo de previsão por redes neurais.

Ao todo são 1453 RPI’s, com extensão “.txt”, publicadas neste período. As bibliotecas Selenium e Beatifusoulp do Python foram utilizadas nesta etapa, para realizar o download automático das RPI’s. As RPI’s ficarão armazenadas na pasta do drive criada no google colab para posterior extração dos dados de interesse.

O programa no Python para download das revistas e demais códigos podem ser visualizados no Apêndice.

O código demorou 38 minutos e 23 segundos para baixar e salvar todas as RPI’s no drive. Após a coleta das RPI’s, precisamos extrair os dados dos arquivos “.txt” armazenados.

Para leitura de cada arquivo precisamos criar uma função modificando o tipo do encoding, pois algumas revistas podem ser lidas com o encoding “utf-8” e outras com “iso-8859-1” sem resultar em erro ao tentar abrir o arquivo. Para realizar a leitura do arquivo, uma função foi desenvolvida, conforme visto no Apêndice.

Com base no conhecimento prévio das RPI’s, foram desenvolvidos três algoritmos de extração (três funções no Python): uma função (pedidos(lista)) para extrair somente as informações dos despachos 1.3, 3.1, 3.2, 2.4 e 23.1.1 retornando um dataframe estruturado com as informações de interesse, uma (exi\_ci(lista)) função para extrair os dados dos despachos 6.1, 6.20, 6.21, 6.22 e 7.1, e uma função (decisão(lista)) para extrair os dados dos despachos 9.1, 9.2 e 11.2 (ver Apêndice).

As três funções foram testadas para extrair os dados das RPI’s 1527, 2080 e 2592, escolhidas aleatoriamente. No teste de extração da RPI 1527, com a função pedidos, falhas foram identificadas após a extração, pois na coluna de idpedido em que deveria haver somente os números dos pedidos, na linha 5 havia trecho de textos. A mesma falha foi identificada na coluna data\_deposito, em que foram encontrados textos em 3 linhas, onde deveria haver somente datas. Não foram identificados erros de extração nas outras colunas.

Observou-se que o erro na extração ocorreu porque em alguns registros, os títulos ou resumos apresentam continuidade de texto nas linhas seguintes, sendo que, coincidentemente, algumas destas continuações se iniciam com caracteres semelhantes a caracteres iniciais que contém o idpedido ou a data\_depósito que se iniciam com (21) e (22), respectivamente. Portanto, para mitigar o erro de extração vamos procurar padrões mais específicos relativos às informações de interesse no sentido de criar um código que possa extrair os registros com acurácia.

Desta forma, no intuito de identificar estes padrões específicos, podemos verificar como aparecem os registros entre parênteses (..) que são números ou letras. O código para nos auxiliar na identificação dos padrões pode ser visto no Apêndice.

Após rodar o programa mencionado acima, foram identificados os padrões mais específicos para a extração das informações, o que ajudará no desenvolvimento do programa de extração com maior acurácia. Os padrões encontrados foram:

1. Em todos os arquivos txt, na linha 0 consta o número da rpi e a data de publicação. Há variações na forma em que a informação foi registrada no arquivo txt, mas a principal é que, em algumas revistas, há muito espaço após a data da RPI. Portanto estes espaços deverão ser removidos para facilitar a extração. A função criada para este propósito foi a remove\_(list) (ver Apêndice).
2. Após a *string* ‘(21) ‘ o pedido geralmente se inicia com letra maiúscula, e pode se iniciar com as letras B, C, M ou P, para pedidos de patentes depositadas (PI-Patente de Invenção, MU-Patente de Modelo de Utilidade, C-Certificado de Adição, BR-Sigla seguida de numeração, fazendo parte da nova numeração dos pedidos de patentes);
3. Após a *string* ‘(22) ‘ a data, obviamente, se inicia com um número;
4. Após a *string* ‘(51) ‘, a classificação geralmente é grafada em letra maiúscula;
5. O título, após a *string* ‘(54) ‘, geralmente se inicia com uma letra maiúscula, ou outro símbolo que não uma letra;
6. A *string* ‘(Cd)’ pode estar grafada também como ‘(CD)’ ou ‘(cd)’. Para facilitar a extração, então, todas estas *strings* serão colocadas em caixa alta aplicando a função c\_alta(list), disponibilizada no Apêndice.

Agora, tomando como base os padrões encontrados na verificação anterior e nas primeiras funções de extração desenvolvidas, foi desenvolvida uma única função para a extração das informações, em que a entrada é uma lista (a RPI transformada em lista) retornando dois dataframes, um contendo os dados extraídos dos despachos 1.3, 3.1, 3.2, 2.4 e 23.1.1e outro contendo os outros despachos. A função de extração final foi definida como pedidos(list) (ver Apêndice).

Realizando os testes de extração nas mesmas RPI’s escolhidas anteriormente, verificou-se que não houve, agora, erros na extração. A primeira vista, a função extraiu as informações com acurácia.

Portanto, podemos seguir para a extração dos dados de todas as RPI’s armazenadas no drive e posterior conversão em uma tabela/dataframe. O código desenvolvido para abrir cada RPI e extrair a informação, linha a linha, utiliza funções, conforme consta no Apêndice.

O tempo de extração dos dados das RPI’s foi de 1 minuto e 22 segundos. Após a extração mais alguns testes foram feitos com os dataframes contendo as informações extraídas. O dataframe (denominado ped\_pub, abreviação para pedidos publicados) contendo os registros dos despachos 1.3, 3.1, 3.2, 2.4 e 23.1.1 possui ao todo 622.407 linhas e 8 colunas (idpedido, data\_deposito, original, ipc, titulo, depositante, despacho, data\_rpi). Já o dataframe com os outros despachos (denominado outros\_total) possui 1.032.349 linhas e 5 colunas (idpedido, data\_deposito, depositante, despacho, data\_rpi). O dataframe (nova\_num\_total) com os registros que informam quais números de pedidos foram alterados e os novos números, contém 4.239 linhas e três colunas (idpedido, novo\_numero e data\_rpi).

Após verificar (xdfadf.sort\_values(by=['qtd\_pub']).head(60)) as revistas que constam 0 (zero) número de pedidos publicados com despachos 1.3, 3.1, 3.2 e 3.4, constatamos que as RPI's 2119 e 2118 possuem sim estes despachos publicados, não nos arquivos txt, mas somente nos arquivos pdf. Com relação às outras RPI's que constam 0 nas publicações, realmente não possuem os despachos citados.

Para extrair os dados dos arquivos pdf (PATENTES2118.pdf e PATENTES2119.pdf) vamos utilizar a biblioteca Apache Tika para a extração de texto de arquivos pdf, pois foi o método que demonstrou melhor extração após um teste rápido com as bibliotecas PyPDF2, Tabula, PDFMiner e o próprio Apache Tika. Com a utilização desta biblioteca teremos uma lista contendo linhas de texto ligeiramente diferente das linhas dos arquivos txt extraídos do website do INPI. Assim, teremos que fazer uma adaptação no código de extração dos dados (ver Apêndice).

O método sort\_values foi aplicado nos dataframes, para cada coluna, e foram visualizados os registros nas 60 primeiras linhas e 60 últimas. Esta simples visualização mostrou diferenças no formato das datas, ipc, título e depositantes. Mesmo com a função customizada para abrir os arquivos e transformá-los em uma lista, não foi possível mitigar os erros na grafia de algumas palavras.

Algumas verificações manuais foram feitas, como nos casos em que aparece o registro ‘NaN’ no lugar da informação. Para isto, foram escolhidos aleatoriamente alguns pedidos e verificou-se que quando é observado o registro ‘NaN’ é porque tal informação realmente não consta na respectiva RPI, ou seja, a função de extração é confiável e extraiu os dados com precisão.

Pelas verificações acima, conclui-se que a etapa de transformação e limpeza dos dados faz-se necessária, devido à discrepância dos registros publicados nas RPI’s. Portanto, com base nas verificações feitas, algumas padronizações serão realizadas na etapa de transformação, tais como:

1. Tratar o dataframe nova\_num\_total no Excel, de forma deixar somente duas colunas (o número antigo e a nova numeração). Alguns ajustes terão que ser feitos manualmente, tendo em vista o fato de que a coluna novo\_numero contém os comentários, os quais possuem o novo número do pedido. Algumas linhas foram excluídas, pois não tratavam de mudança de numeração ou era a renumeração de pedidos pipelines, que não serão estudados no presente trabalho. Este tratamento será feito no excel, dado que não há uma padronização nas alterações dos números dos pedidos, sendo que muitos passos terão que ser feitos manualmente;
2. Criar uma nova coluna, que será a coluna chave, a qual será denominada de chave\_idpedido. Nesta coluna serão removidas todas as *strings*, inclusive, após o traço “-“. A coluna idpedido será mantida;
3. Padronizar as colunas da data de depósito e data da RPI nos dois *dataframes*, pois estas datas possuíam diversos formatos nas RPI’s: dd/mm/yyyy, dd/mm/yy ou ddmmyy. O formato padrão escolhido foi yyyy-mm-dd;
4. Criar uma coluna contendo somente a primeira ipc do pedido no dataframe “ped\_pub” (sem o ano de atualização da IPC, informação que consta nas revistas mais recentes, entre parênteses) para simplificação de análise, pois há pedidos que chegam a possuir mais de cinco classificações IPC;
5. Excluir os caracteres especiais (#$&? etc.) dos títulos e passar todos os caracteres dos títulos para caixa alta no *dataframe* “ped\_pub”;
6. Criar uma coluna contendo somente o primeiro depositante e passando os caracteres para caixa alta nos dois *dataframes*;
7. Criar mais três colunas subdividindo as classificações IPC: ipc\_1 contém a seção da IPC (primeira letra da IPC), ipc\_2 contém a subseção da ipc (três primeiros caracteres da IPC) e ipc\_3 contém a classe da IPC (quatro primeiros caracteres da IPC) no dataframe “ped\_pub”. Em suma, quanto maior a subdivisão, maior o detalhamento técnico informado pela IPC;
8. Criar uma coluna informando o país de origem do depositante e outra contendo o estado, caso o país depositante seja o Brasil (BR);
9. Verificar as repetições na coluna “idpedido” do dataframe “ped\_pub”, e excluir as repetições mantendo-se o registro que contém a publicação na RPI mais recente;
10. Criar uma nova coluna contendo a descrição da área e campo tecnológicos de acordo com a classificação IPC, segundo a OMPI – Organização Mundial da Propriedade Intelectual (https://www.wipo.int/ipstats/en/), utilizando o método merge;
11. Criar uma nova coluna informando se o depositante é Residente (Brasil) ou Não-Residente (Internacional);
12. Juntar os dois dataframes utilizando o método merge, e criar um dataframe único contendo os dados bibliográficos dos pedidos de patentes publicados e as ações posteriores que estes pedidos sofreram ao longo dos anos pelo INPI. No entanto, este dataframe será utilizado apenas para uso interno do INPI;
13. Padronizar ao menos 150 nomes de depositantes. Esta padronização teve que ser feita no excel.
14. Após todos os passos anteriores, no próprio excel, os dados foram compilados em dataframes menores, contendo as contagens. O método utilizado será a “tabela dinâmica”.

Detalhes dos códigos utilizados na transformação podem ser visualizados no Apêndice.

Os arquivos com os dados compilados são:

1 - despachosxano.csv: contém os despachos de maior interesse para a diretoria de patentes (DIRPA), pois são os despachos em que se afere a produção desta diretoria. Os despachos referem-se às decisões técnicas com relação aos pedidos de patentes publicados (Deferimento – 9.1, Indeferimento – 9.1, Arquivamento técnico - 11.2 e Arquivamento técnico – 11.5). A métrica utilizada para fins de avaliação anual é a soma da quantidade destes despachos anualmente.

2 - res\_nao\_resxano.csv: contém o número de pedidos de patente dos depositantes residentes e não residentes ano a ano.

3 - paisxano.csv: contém o número de pedidos de patente por país, anualmente.

4 - depositantexano.csv: contém o número de pedidos de patentes por depositante, anualmente.

5 - estadoxano.csv: contém o número de pedidos de patentes por estados brasileiros, ano a ano.

6 - depositante\_residentexano.csv: contém o número de pedidos de patentes dos depositantes residentes no Brasil.

7 - ompi\_USxano.csv: contém o número de pedidos de patentes dos Estados Unidos por campo e área tecnológicos, ano a ano.

8 - ompi\_BRxano.csv: contém o número de pedidos de patentes do Brasil por campo e área tecnológicos, ano a ano.

9 – pub\_mes.csv, df\_test.csv e df\_train.csv: contém o número de pedidos de patente geral mês a mês.

O código base utilizado para gerar os gráficos com animação (Bar Chart Race) pode ser visualizado no Apêndice. Este código teve como base o código publicado por (Vardhan, 2019).

Será utilizado um modelo de rede neural simples, com uma camada apenas contendo 20 neurônios somente, utilizando as bibliotecas Keras e Sklearn (Gulli & Pal, 2017). Assim vamos rodar o código original com a base de dados, ver os resultados e fazer as seguintes mudanças:

1 – Mudar o otimizador de adam para sgd dentro da configuração da rede e ver os resultados de RMSE;

2 – Voltando a configuração inicial mudar o número de neurônios da camada LSTM, primeiro para 10 e depois para 30 e ver o resultado do RMSE;

3 – Colocar uma segunda camada LSTM, contendo 5, 10 e 15 neurônios e ver o resultado do RMSE;

4 – Mudar por dois valores que achar conveniente às épocas de treinamento e ver o resultado do RMSE;

5 – Com as mudanças feitas, escolher a configuração que apresentar o menor RMSE.

Com os testes acima vamos preencher a seguinte tabela:

|  |  |  |  |
| --- | --- | --- | --- |
| Código | RMSE (base treino) | RMSE (base teste) | “Early Stopping” |
| Inicial |  |  |  |
| Otimizador sgd |  |  |  |
| 10 neurônios |  |  |  |
| 30 neurônios |  |  |  |
| Com 2º camada (XX neurônios) |  |  |  |
| Época limitada até XXX |  |  |  |
| Época limitada para XXX |  |  |  |

O código base para gerar a rede neural e treinar esta rede é mostrado no Apêndice. Este código é de autoria do professor Leonardo Mendonza, do departamento de Engenharia Elétrica da PUC-Rio. Para gerar a rede e treinar a mesma, o código foi dividido em 4 partes, a saber:

Parte 1 – Pre-processamento: Nesta etapa os dados serão divididos em base de teste e base de treino. A base para treino terá 67% dos dados (período de 1993 a 2013) e a base teste para a rede treinada terá 33% dos dados (2014 a 2016). O valor do RMSE será calculado tanto para a base de teste quanto para a base de treino.

Parte 2 – Construção da RNN

Parte 3 – Previsões e análise dos resultados

Parte 4 – Métricas de avaliação (RMSE-Root Mean Square Error).

* 1. RESULTADOS

A visualização gráfica do arquivo despachosxano.csv pode ser vista na figura 9.

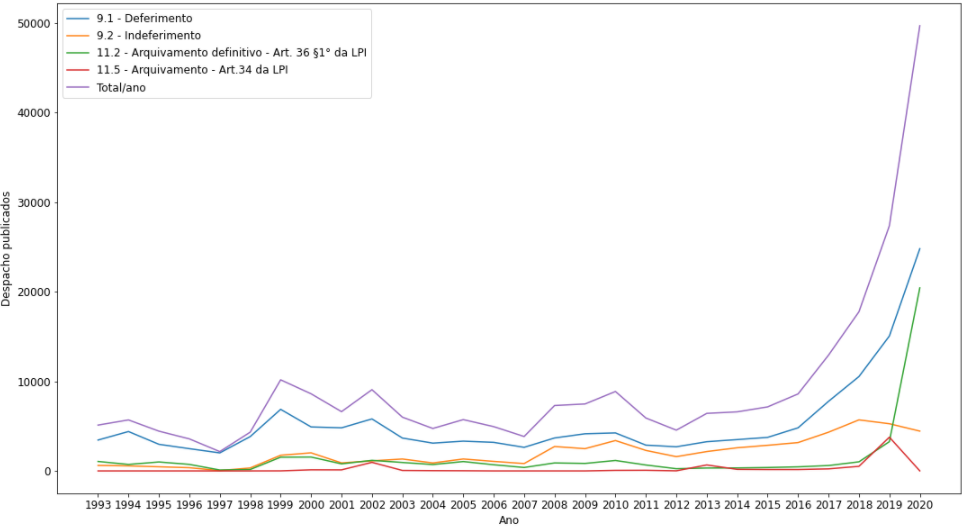


Figura 9 – Despachos de decisão da Diretoria de Patentes do INPI (1993 a 2020).

Pode-se notar que o número total de despachos, ditos despachos de decisão (9.1 - Deferimento, 9.2 - Indeferimento, 11.2 - Arquivamento definitivo e 11.5 - Arquivamento) começaram a ter um aumento significativo a partir de 2016. O que evidencia um aumento significativo no número de decisões da Diretoria de Patentes. Uma das razões para este aumento na produtividade da DIRPA tem relação direta com o plano de ataque ao BACKLOG do INPI, bem como ao aumento significativo na quantidade de pedidos de patentes analisados pelos examinadores de patentes do Instituto.

A figura 10 mostra os dados contidos no arquivo res\_nao\_resxano.csv.

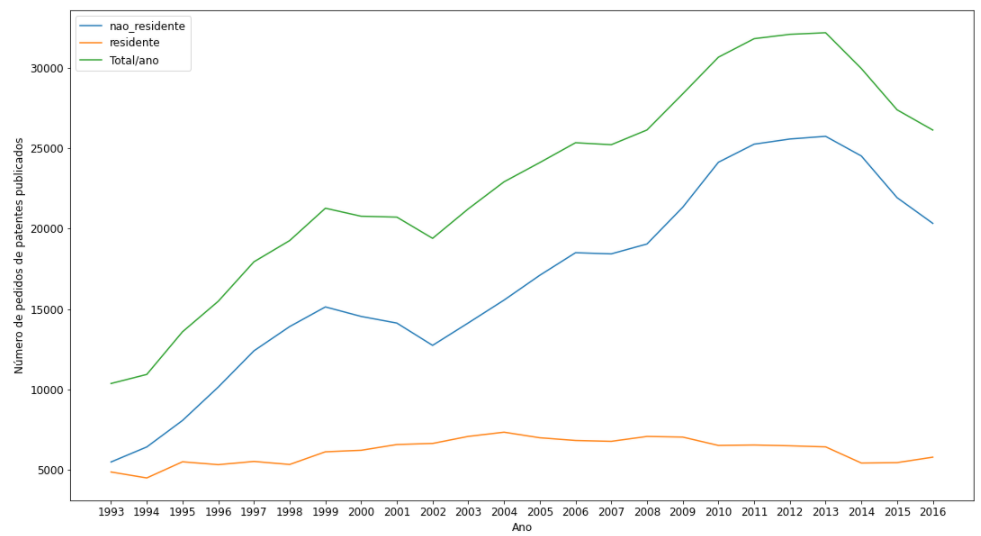


Figura 10 – Número de pedidos de patentes publicados por ano (1993 a 2016)

Pela evolução do gráfico, é notória a diferença entre o depósito de pedidos de patentes nacional e internacional. Observa-se que entre 1993 e 1995 a diferença entre os depósitos nacionais e internacionais era bem menor quando esta diferença é comparada, por exemplo, em 2013.

Inúmeras explicações podem ser dadas a este fato. Fatores sócio-econômicos, investimento em ciência/inovação e tratados internacionais podem ser citados para explicar tal discrepância, mas esta explicação foge do escopo do presente trabalho.

A partir dos dados da figura 10, algumas questões podem ser levantadas, tais como: Dentre os depositantes, quais os países com maior número de pedidos ao longo dos anos? Quais empresas possuem maior número de pedido de patentes?

As figuras 11 (visualização gráfica do arquivo paisxano.csv) e 12 (visualização gráfica do arquivo depositantexano.csv) são gráficos estáticos, do ano de 2016, que mostram o número de depósitos de pedidos de patentes no Brasil, por país (figura 11) e por empresa (figura 12) e agrupados (pelas cores) por continente. Nestas figuras, teremos as respostas para as questões acima.

Os vídeos 1 e 2 mostram a evolução ano a ano, mostrando qual país (vídeo 1) e qual empresa (vídeo 2) lidera o número de depósito de pedidos de patentes no Brasil.

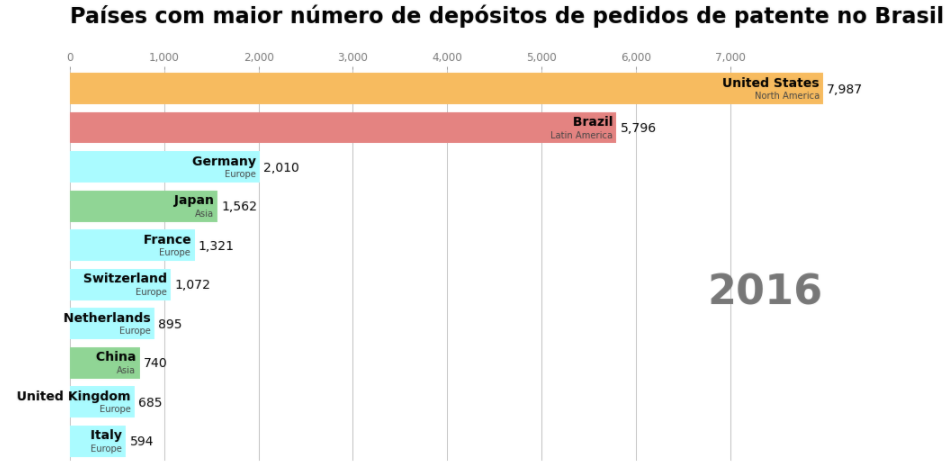


Figura 11 – Países com maior número de pedidos de patentes no Brasil – 2016 (ver vídeo pais\_ano.mp4 abaixo para observar a evolução de 1993 a 2016.



Vídeo 1 – Evolução do número de pedidos de patentes ano a ano, por país.

Pela visualização do vídeo 1, podemos observar que no período de 1993 a 1997 o Brasil era o que mantinha o maior número de depósitos de pedidos de patente em seu território. Em 1998 e 1999 os Estados Unidos assumiram a liderança e, de 1999 a 2005, o Brasil voltou a ser o primeiro em número de depósitos. Após 2005, os Estados Unidos seguiram como os maiores depositantes em território brasileiro, se afastando cada vez dos depositantes nacionais, e de outros países, até 2016. Como pode ser visto na figura 11. Também é notório que o continente europeu é maioria entre os 10 maiores depositantes, independente do ano. O Japão seguia isolado como representante asiático até o ano de 2012, sendo que a partir de 2013 a China entra para seleto grupo dos top 10.

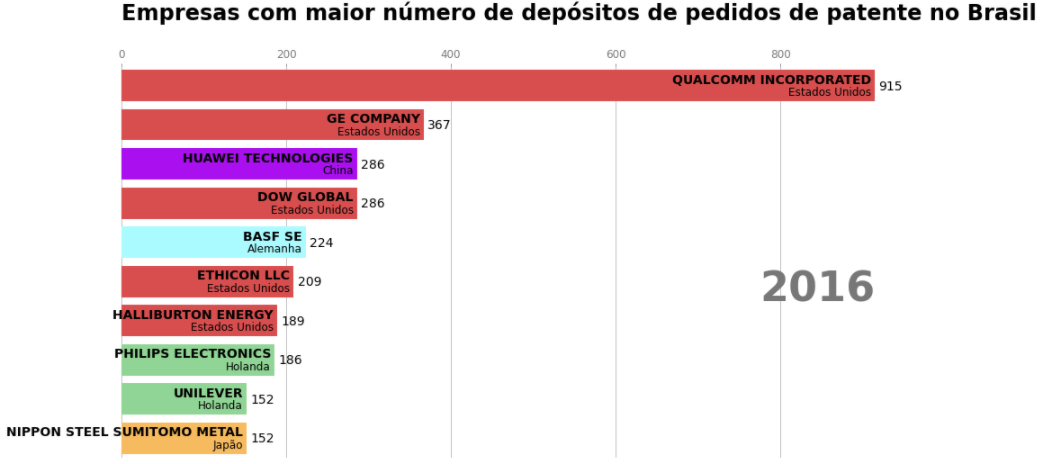


Figura 12 – Empresas com maior número de depósitos de pedidos de patente no Brasil (2016)



Vídeo 2 – Evolução do número de pedidos de patentes ano a ano, por empresa depositante.

Com a visualização do vídeo 2, observa-se que a empresa com maior número de pedidos de patentes no Brasil, no período de 1993 a 2000 foi a Procter & Gamble, empresa americana de bens consumo que abrange diversos produtos, dentre os quais se destacam os de higiene pessoal.

Em 2001 foi a vez da Bosch ficar em primeiro lugar no depósito de pedidos de patentes no Brasil, empresa notadamente conhecida por soluções automotivas e máquinas ferramentas. Nos anos de 2002, 2003, 2006 a 2009 e 2014 a 2016, a Qualcomm dominou o primeiro lugar no depósito de pedidos de patentes, isto é, nove anos de domínio de inovação no país. A Qualcomm é uma empresa destacadamente conhecida no ramo da telecomunicação.

Em 2004 e 2005 o destaque foi para as companhias Basf e Microsoft. A Basf, empresa bastante conhecida na área química, atua em diversas áreas, fornecendo soluções diversas para inúmeras áreas. A Microsoft dispensa comentários. Já no período de 2010 a 2013 a Philips emplacou o primeiro lugar no depósito de pedidos de patentes, empresa conhecida por produtos eletrônicos voltados para a área de higiene pessoal, setor automotivo etc.

Já, com relação ao Brasil, pode-se questionar: qual é o estado com maior número de depósitos de pedidos de patentes? Existem uma ou duas regiões predominantes? Na figura 13 e nos vídeos 3 e 4 teremos as respostas. Os estados foram agrupados por região (cada região possui uma cor).

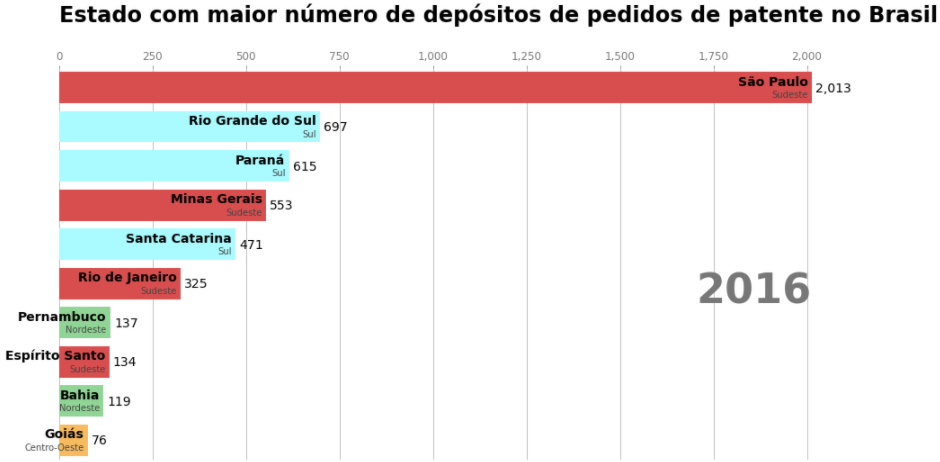


Figura 13 – Estados com maior número de depósitos de pedidos de patente (2016)



Vídeo 3 – Evolução do número de pedidos de patentes ano a ano, por estado brasileiro.

É notória a hegemonia das regiões sudeste e sul no cenário de inovação brasileiro. Sendo os estados que se destacam: São Paulo (desde 1993 está em primeiro lugar nos top 10), Rio de Janeiro, Rio Grande do Sul, Minas Gerais, Paraná e Santa Catarina.

Estas informações são consistentes, pois como citado em estudos anteriores, há uma forte correlação entre o PIB e o número de depósitos de pedidos de patentes. Não por acaso, estes estados ocupam os 6 primeiros lugares desde 1993 no número de depósitos de pedidos de patentes, pois são os estados com o maior PIB do Brasil, conforme levantamento do IBGE (https://www.ibge.gov.br/explica/pib.php).

Outra maneira, visualmente mais elegante, de ver estes dados, seria plotando em um mapa, com cores, como no arquivo abaixo:



Vídeo 4 – Estados com maior número de depósitos de pedidos de patentes (1993 a 2016)

O vídeo 4 aparenta não mostrar diferença ao logo dos anos, o que de fato acontece, pois São Paulo é o estado com maior número de pedidos de patentes, se destacando dos outros, não importa o período. O código utilizado para criar o vídeo 4 foi inspirado no código publicado por (Maia, 2020) e não será exibido neste trabalho, pois foi realizado apenas um teste para mudar a visualização dos dados.

Com os dados já mostrados, sabemos quais as empresas, de um modo geral, tem maior número de pedidos de patentes no Brasil. No entanto, ainda não sabemos quais as empresas com residência no Brasil possuem maior número de pedidos de patentes no INPI.

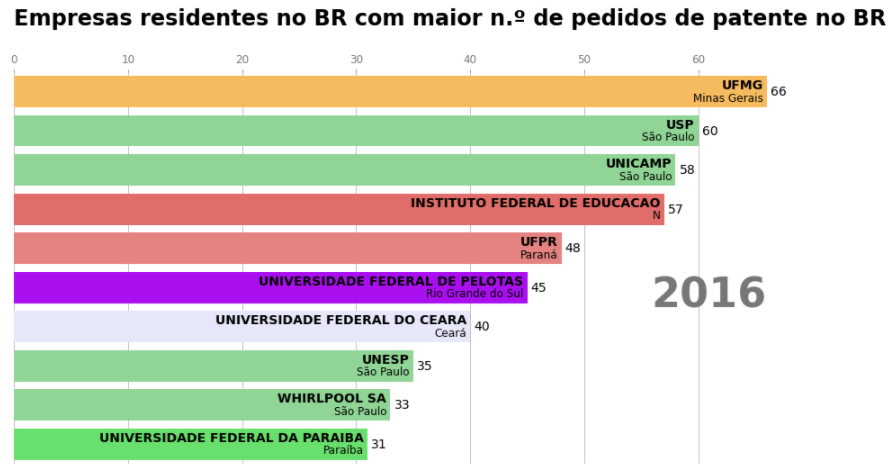


Figura 14 – Empresas residentes no Brasil com maior número de pedidos de patentes no INPI (2016)



Vídeo 5 – Empresas residentes no Brasil com maior número de pedidos de patentes no INPI (1993 a 2016)

A figura 14 e o vídeo 5 mostram quais as empresas/universidades com residência no Brasil possuem o maior número de pedidos de patentes no INPI ao longo dos anos, dentre as quais, se destacam a Petrobrás, CSN, Cia Vale do Rio Doce, Usiminas, Semato, Whirlpool e tantas outras. Além destas empresas, as universidades públicas também estão entres os Top 10 ao longo dos anos, cito: Unicamp, UFMG, USP, UFRJ, UFPR, UFRGS dentre outras federais. Interesse notar que em 2016, dos 10 maiores depositantes, 9 são universidades públicas e não empresas.

Vimos na figura 11 e no vídeo 1 que os dois países com maior número de pedidos de patentes no INPI são os Estados Unidos e o próprio Brasil. Podemos ir um pouco mais a fundo nestes dois países e questionar, por exemplo, quais os campos e áreas tecnológicas de maior investimento dos mesmos. As figuras 15 e 16 e os vídeos 6 e 7 trarão mais detalhes.

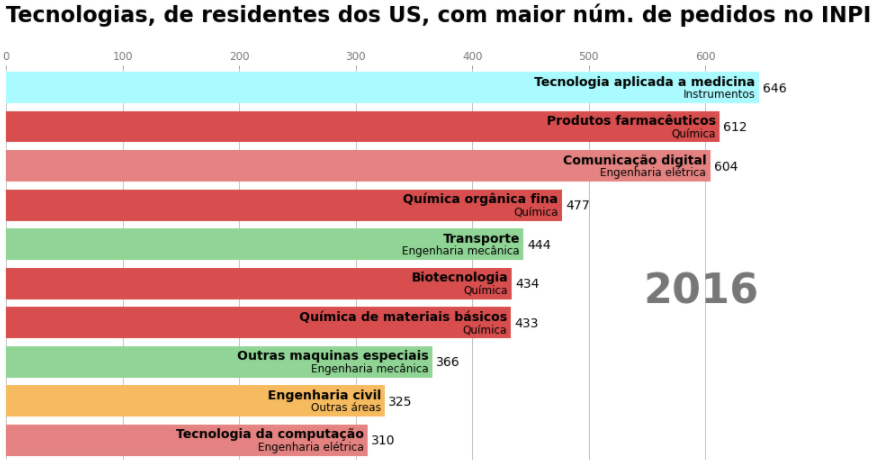


Figura 15 – Tecnologias, com residentes nos Estados Unidos, com maior número de pedidos de patentes no INPI (2016)



Vídeo 6 – Tecnologias, com residentes nos Estados Unidos, com maior número de pedidos de patentes no INPI (1993 a 2016)

Observa-se que na década de 1990 até 2010, a área tecnológica da química ocupou os três primeiros lugares, seguida da área de instrumentos, engenharia mecânica e engenharia elétrica, tendo a última, destaque para os campos tecnológicos das telecomunicações e computação. A partir de 2011 as áreas tecnológicas com maior destaque foram: Instrumentos, Engenharia civil e Engenharia elétrica.

É importante frisar que os produtos da área química, tais como produtos farmacêuticos, bem como a tecnologias aplicadas à medicina, possuem grande valor agregado, principalmente no que tange à sua necessidade, sendo evidenciada pela atual crise sanitária, provocada pelo coronavirus, a qual demonstrou a importância dos respiradores mecânicos, máscaras e vacinas.

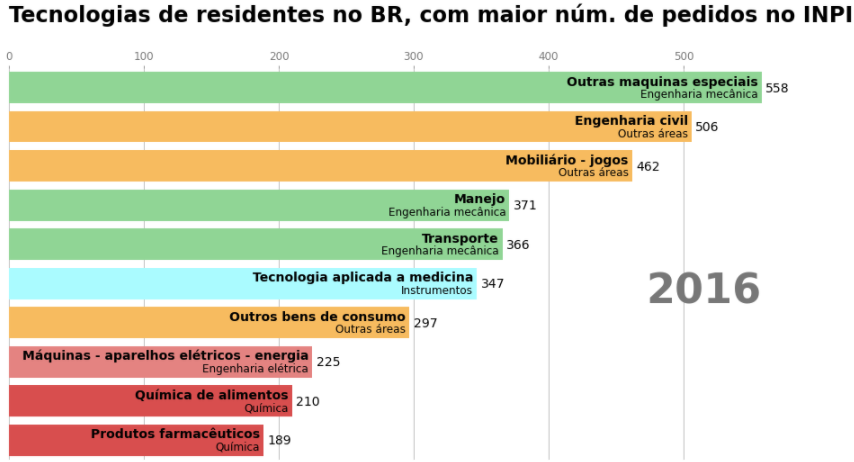


Figura 16 – Tecnologias, com residentes no Brasil, com maior número de pedidos de patentes no INPI (2016)



Vídeo 7 – Tecnologias, com residentes no Brasil, com maior número de pedidos de patentes no INPI (2016)

No vídeo 7, verifica-se que na década de 1990 até 2009, o campo tecnológico com mais depósitos de pedidos de patentes, por parte de depositantes residentes no Brasil, foi o ramo de mobiliário-jogos (outras áreas), seguida de engenharia civil (outras áreas), bem como o campo de transporte e manejo (ambos da área de engenharia mecânica). A partir de 2010 até 2016 o campo tecnológico de engenharia civil (outras áreas) e outras máquinas especiais (engenharia mecânica) disputaram o primeiro lugar.

Cabe mencionar que os campos tecnológicos de depositantes nacionais mencionados acima possuem menor valor agregado quando comparados aos campos tecnológicos de depositantes residentes nos Estados Unidos.

Os gráficos e vídeos mostrados até aqui foram uma pequena amostra do que é possível fazer com os dados coletados, em termos de Business Intelligence. Muito mais poderia ter sido mostrado, como análises diretas a partir das IPC’s (classificação internacional de patentes) ou mesmo análises de NLP a partir dos títulos dos pedidos de patentes.

Vamos agora para os resultados para a previsão dos depósitos de pedidos de patentes.

**Resultados para o código original**

A série a ser prevista pode ser vista na figura 17. Observando-se o comportamento dos dados ao longo do tempo, vê-se que é uma função bastante próxima de uma função oscilatória com vales e picos variáveis e bastante acentuados.

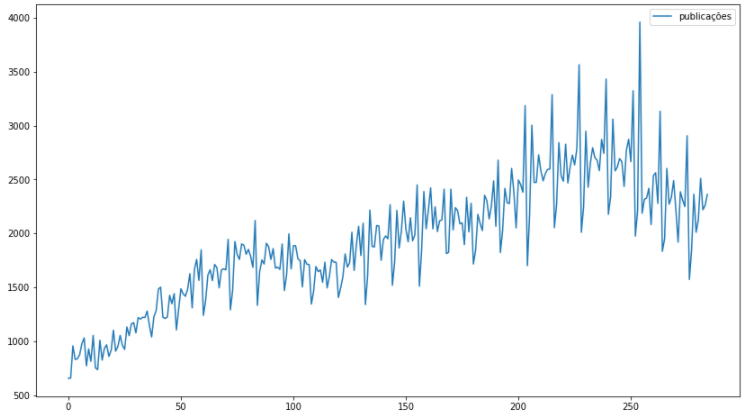


Figura 17 – Dados ao longo do tempo

Ao treinar o modelo, nota-se que o número de épocas não passa de 340, onde há uma parada prematura (“early stopping”). As figuras 18 e 19 mostram até que época ocorre o treinamento. Interessante notar que há uma tendência da linha azul (val\_loss - validação) se aproximar da linha laranja (loss – erro do treinamento), mas não ao ponto de tornar a rede especializada na base de treino, indicando uma boa capacidade de generalização.

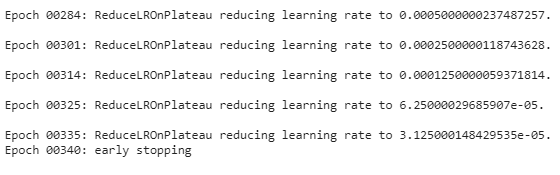


Figura 18 – Número de épocas para o treinamento no código original

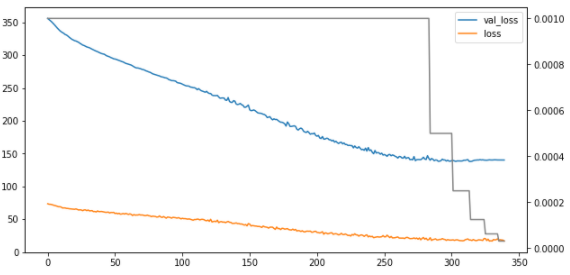


Figura 19 – Treinamento da época 0 à última

Os valores RMSE para a base de treinamento e base de teste são, respectivamente: 4,04 e 11,84.

Observa-se que, de modo geral, a previsão com o código original é boa, como pode ser visualizado na figura 20. A linha laranja é a previsão da base de treino e a linha verde é a previsão para a base teste, a linha azul, obviamente, é dos valores reais.

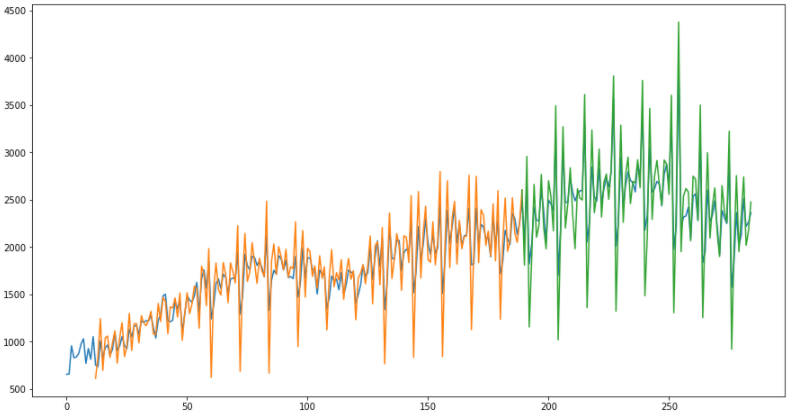


Figura 20 – Previsão para a base teste e base de treino

**Resultados para o otimizador igual a sgd**

Quando mudamos o otimizador de “adam” para “sgd” observamos que o número de épocas não passa de 211, onde ocorre a “parada prematura”. A figura 21 mostra quando ocorre a parada prematura.

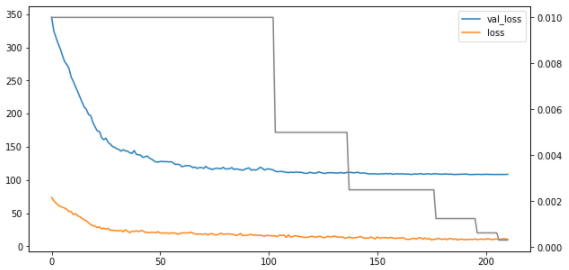


Figura 21 - Treinamento da época 0 à última

Os valores dos RMSE para a base de treinamento e teste foram 2,92 e 10,42, valores estes menores que o mostrado no código original. Desta forma, com base somente no RMSE (quanto menor, melhor a previsão), infere-se que a previsão utilizando o otimizador sgd foi melhor que a previsão utilizando o otimizador adam.

**Resultados para número de neurônios igual a 10 e 30**

Retornando o código com o otimizador “adam” e diminuindo o número de neurônios para 10, observamos que o treinamento ocorreu até 675 (ver figura 22) épocas e os valores do RMSE para a base de treino e teste foram 3,91 e 12,24, respectivamente. Nota-se que houve mais épocas no treinamento e o valor do RMSE foi um pouco maior do que a mudança do otimizador para “sgd”. Com 10 neurônios a previsão também não se mostrou ruim.

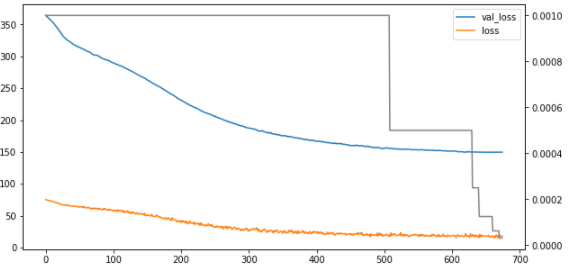


Figura 22 - Treinamento da época 0 à última, para 10 neurônios

Já para 30 neurônios houve treinamento até 139 épocas (ver figura 23) e os valores do RMSE foram de 6,58 (base de treino) e 14,92 (base de teste).

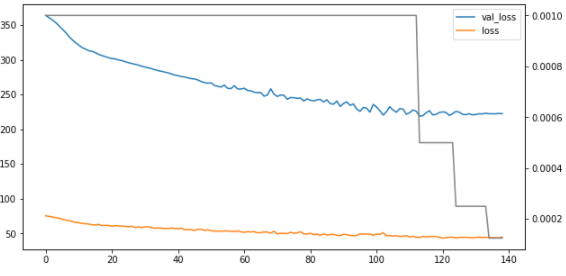


Figura 23 - Treinamento da época 0 à última, para 30 neurônios

**Resultados para o modelo com a adição de uma camada**

Uma primeira dificuldade para inserir a segunda camada seria a de saber quantos neurônios deve ter esta 2º camada. Vamos avaliar três resultados para os seguintes números de neurônios, escolhidos aleatoriamente: 5, 8 e 12. A tabela 1 mostra os resultados.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Neurônios na segunda camada** | | |
| **Modelos** | 5 | 8 | 12 |
| **Treino** | 5,54 | 4,56 | 6,32 |
| **Teste** | 15,13 | 13,84 | 14,69 |
| **Early stopping** | 354 | 413 | 218 |

Tabela 1 – Valores do RMSE e Épocas para cada quantidade de neurônios da 2º camada.

Pela tabela 1 verifica-se que o menor RMSE (4,56 e 13,84) foi para a 2º camada com número de neurônios igual a 8. Portanto, a melhor previsão (menor RMSE) é para o número de neurônios igual a 8 na segunda camada.

**Resultados para o modelo mudando os valores para a época**

Observou-se que, para todas as configurações anteriores, antes de ocorrer o “early stopping”, há um rápido decréscimo da taxa de aprendizado (linha cinza), o que não é recomendável, pois a rede pode estar se especializando em um mínimo local. Com base nesta observação, podemos limitar o número de épocas para um valor imediatamente anterior à queda da taxa de aprendizado e verificar o RMSE. Após alguns testes, verificou-se que o valor imediatamente anterior à queda da taxa de aprendizado é 235.

Podemos escolher então o valor de 235 e um valor um pouco depois da queda de taxa de aprendizado, como 245, por exemplo.

Portanto, para 235 épocas o valor do RMSE foi de 4,45 (base treino) e 12,28 (base teste). Já para 245 épocas o RMSE foi de 5,21 (base treino) e 13,61 (base teste). Observa-se, portanto, que houve uma redução no RMSE para a base teste com 220 épocas de treinamento, quando comparado ao valor de 245 épocas de treinamento.

**Escolha da melhor configuração**

Para escolher a melhor configuração com base nas mudanças realizadas no código, vamos comparar os valores do RMSE para cada mudança, colocando-os em uma tabela (tabela 2).

|  |  |  |  |
| --- | --- | --- | --- |
| **Código** | **RMSE (base treino)** | **RMSE (base teste)** | **“Early Stopping”** |
| **Inicial** | 4,04 | 11,84 | 340 |
| **Otimizador sgd** | 2,92 | 10,42 | 210 |
| **10 neurônios** | 3,91 | 12,24 | 680 |
| **30 neurônios** | 6,58 | 14,92 | 140 |
| **Com 2º camada (8 neurônios)** | 4,56 | 13,84 | 413 |
| **Época limitada até 235** | 4,45 | 12,28 | -- |
| **Época limitada para 145** | 5,21 | 13,61 | -- |

Tabela 2 – RMSE para cada modificação no código

Pela tabela 2 é possível verificar que o menor RMSE foi quando utilizamos o otimizador sgd, ao invés do adam.

Desta forma, podemos inferir que, a partir das configurações acima, a melhor previsão é com o otimizador sgd, pois o mesmo foi o que apresentou menor RMSE.

No entanto, tal como apontado anteriormente, antes do “early stopping” há uma redução rápida da taxa de aprendizado, e verificou-se também que houve uma melhora no valor do RMSE ao se limitar o valor das épocas para um número anterior à queda da taxa de aprendizado.

Em face dos resultados acima, pode-se perguntar se é possível obter um resultado melhor para os valores dos RMSE. Vamos aos testes. Como o melhor resultado do RMSE foi para o otimizador sgd, podemos manter este otimizador e mudar os outros parâmetros tal como foi feito para o código original, limitando agora o número de épocas para um valor anterior à redução da taxa de aprendizado ou para antes do overfitting, e comparar os valores do RMSE.

|  |  |  |  |
| --- | --- | --- | --- |
| **Código com otimizador sgd** | **RMSE (base treino)** | **RMSE (base teste)** | **Época** |
| **Camada única com 10 neurônios** | 4,13 | 11,35 | 105 |
| **Camada única com 30 neurônios** | 4,03 | 10,54 | 80 |
| **Camada única com 50 neurônios** | 4,01 | 10,15 | 65 |
| **Camada única com 70 neurônios** | 4,11 | 9,98 | 70 |
| **Camada única com 100 neurônios** | 4,21 | 9,85 | 60 |
| **Camada única com 200 neurônios** | 4,31 | 9,44 | 45 |
| **Camada única com 500 neurônios** | 4,29 | 8,72 | 60 |
| **Camada única com 800 neurônios** | 4,35 | 8,58 | 50 |
| **Camada única com 1000 neurônios** | 4,31 | 8,45 | 45 |
| **1º camada com 500 neurônios**  **e 2º camada (100 neurônios)** | 4,31 | 10,88 | 60 |
| **1º camada com 500 neurônios**  **e 2º camada (200 neurônios)** | 4,46 | 10,16 | 45 |
| **1º camada com 500 neurônios**  **e 2º camada (300 neurônios)** | 4,43 | 9,95 | 45 |

Tabela 3 – RMSE para otimizador igual a sgd e épocas limitadas ao valor anterior à redução da taxa de aprendizado

Observamos que, à medida que aumentamos o número de neurônios ao modelo de camada única, há uma sensível diminuição no valor do RMSE para a base de teste. No entanto, a partir de 500 neurônios a redução pode não ser considerada significativa, e, além disso, quanto maior o número de neurônios, maior é o tempo para executar o treinamento da base de treino, podendo não ser ideal adotar o modelo com 1000 neurônios, mas sim com 500 neurônios.

Foi verificado, também, se havia alguma melhora no valor do RMSE caso fosse adicionado uma segunda camada, fixando a primeira camada com 500 neurônios e variando a 2º camada para 100, 200 e 300 neurônios. No entanto, não houve redução significativa.

Portanto, pelos resultados visualizados na tabela 3, podemos adotar a modelagem de apenas uma camada com 500 neurônios e número de épocas igual a 60, sendo que 60 épocas é o valor em que não ocorre tendência de “overfitting” e não ocorre o rápido decaimento da taxa de aprendizagem, como pode ser visto na figura 24.

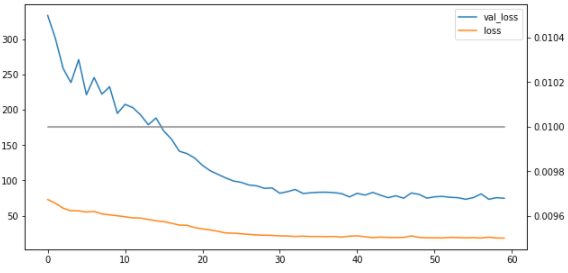


Figura 24 - Treinamento da época 0 à 60 com otimizador igual a sgd e camada única com 500 neurônios

O código da rede neural adotada pode ser vista no Apêndice.

* 1. Conclusões e TRABALHOS FUTUROS

Com este trabalho foi possível demonstrar que os dados disponíveis nas RPI’s (revista da propriedade industrial) podem ser utilizados para rastrear tecnologias de grandes empresas, países e estados brasileiros. Isto é, são dados relevantes e totalmente públicos e disponíveis para quem tiver interesse.

A fase de tratamento dos dados, como era de se esperar, compôs pelo menos 60% do tempo de toda a análise. Após a fase de tratamento dos dados, o restante do processo fluiu um pouco mais rápido.

O maior desafio foi o de desenvolver um código para extrair os dados de interesse dos arquivos txt (ou pdf) e transformar estes dados em um dataframe passível de análise. A fonte de inspiração para criar o algoritmo foram as técnicas de webscraping. Com o algoritmo de extração foi possível extrair das RPI’s o número de pedidos de patentes depositados publicados no Brasil, as tecnologias com maior número de depósitos no país, os países com maior número de pedidos de patentes no INPI, os estados brasileiros que possuem maior número de depósitos bem como os despachos relevantes emitidos pelo INPI, bem como obter o número de pedidos de patentes depositados mês a mês, desde janeiro de 1993 a dezembro de 2016.

Com o histórico mensal de pedidos de patentes foi possível treinar a rede neural modelada e a partir desta rede é possível prever um valor provável de pedidos de patentes, o que, em teoria, pode guiar o INPI no planejamento da necessidade de contratação, metas de produção e demais recursos necessários para melhorar a eficiência da Autarquia.

Com relação à escolha do melhor modelo para previsão temporal, infere-se que, com base nos testes realizados, é possível melhorar a previsão alterando-se alguns parâmetros do modelo, tais como: otimizador, número de neurônios da camada única, adicionar ou não uma 2º camada à rede, ou mesmo modificar o número de épocas de treinamento, tendo em vista que não é recomendável o overfitting e/ou rápido decréscimo da taxa de aprendizagem.

O depósito de pedidos de patentes, por depender de diversos fatores sócio-econômicos, obviamente, torna-se um parâmetro cuja incerteza pode afetar a previsão. Desta forma, como um trabalho futuro, diversas simulações podem ser realizadas utilizando-se o método de Monte Carlo, estimando valores médios, mínimos e máximos ao longo do tempo.

Outra análise, de forma específica, pode ser realizada escolhendo-se um campo tecnológico e aprofundar o estudo de tendência desta tecnologia, especificamente, tais como fármacos, equipamentos de aplicação na medicina e/ou tecnologias desenvolvidas para o consumo sustentável etc. E a partir deste aprofundamento, utilizar redes neurais ou simulação de Monte Carlo para prever cenários diversos.

**REFERÊNCIAS BIBLIOGRÁFICAS**

Abbas, A., Zhang, L., & Khan, S. (Junho de 2014). A literature review on the state-of-the-art in patent analysis. *World Patent Information*, pp. 3-13.

Aristodemou, L., & Tietze, F. (Dezembro de 2018). The state-of-the-art on Intellectual Property Analytics (IPA): A literature review on artificial intelligence, machine learning and deep learning methods for analysing intellectual property (IP) data. *World Patent Information*, pp. 37-51.

Buainain, A., & Souza, R. (2018). *Propriedade intelectual, inovação e desenvolvimento: desafios para o Brasil.* Rio de Janeiro: ABPI.

*Cartilha para desenvolvedores, dados abertos. Governo Federal.* (2011). Acesso em 2020, disponível em W3C: https://www.w3c.br/pub/Materiais/PublicacoesW3C/manual\_dados\_abertos\_desenvolvedores\_web.pdf

Ciancio, A., Santos, C., Freeland, F., Lacerda, M., & Latsch, V. (2017). *Criação de uma máquina de estados para os processos de patentes do INPI.* Rio de Janeiro.

Di Petta, A., & Ferraz, R. (2020). Procurando por inovação? Que tal usar uma ferramenta gratuita para procurar em 110 milhões de patentes? In: M. Machado, *Diálogo conceitual e metodológico das ciências sociais aplicadas com outras áreas do conhecimento* (pp. 160-179). Ponta Grossa: Atena.

Gulli, A., & Pal, S. (2017). *Deep learning with keras.* Birmingham: Packt Publishing.

Hidalgo, A., & Gabaly, S. (Março de 2012). Use of prediction methods for patent and trademark applications in Spain. *World Patent Information*, pp. 19-29.

Hingley, P., & Park, W. (Setembro de 2015). A dynamic log-linear regression model to forecast numbers of future filings at the European Patent Office. *World Patent Information*, pp. 19-27.

Kuotsan, L., & Chen, Y. (2014). A study of patent numbers forecasting by linear regression on cloud. *International Journal of Arts and Commerce*, 207-217.

Macedo, M., & Barbosa, A. (2000). *Patentes, pesquisa & desenvolvimento: um manual de propriedade intelectual.* Rio de Janeiro: FIOCRUZ.

Maia, N. (17 de Novembro de 2020). *https://python.plainenglish.io/*. Acesso em 25 de Março de 2021, disponível em Python in plain englis: https://python.plainenglish.io/how-to-create-a-interative-map-using-plotly-express-geojson-to-brazil-in-python-fb5527ae38fc

Markovic, D., Petkovic, D., Nikolic, V., & Milovancevic, M. (Agosto de 2017). Determination of important parameters for patent applications. *Determination of important parameters for patent applications*. Facta Universitatis.

Oliveira, L., Suster, R., Pinto, A., Ribeiro, N., & Silva, R. (2005). Informação de patentes: ferramenta indispensável para a pesquisa e o desenvolvimento tecnológico. *Quim. Nova*, 5.

Paranaguá, P., & Reis, R. (2009). *Patentes e Criações Industriais.* Rio de Janeiro: FGV.

Pereira, D. (2018). Uma análise do Instituto Nacional de Propriedade Industrial quanto aos registros de marcas. *Uma análise do Instituto Nacional de Propriedade Industrial quanto aos registros de marcas*. Rio de Janeiro, RJ, Brasil: FGV.

Saad, M., Jneid, M., & Saleh, I. (2020). Predicting technology success based on patent data, using a wide and deep neural network and a recurrent neural network. Granada, Spain.

Silva, M. d., & Silva, J. (2013). Um estudo comparativo entre a legislação francesa e brasileira referente à proteção da propriedade intelectual, inovação e seu reflexo no desenvolvimento destas nações. *Revista Jurídica ESMP*, 24.

Vardhan, P. (06 de Setembro de 2019). *Towards Data Science*. Acesso em 20 de Feveiro de 2021, disponível em Towards Data Science: https://towardsdatascience.com/bar-chart-race-in-python-with-matplotlib-8e687a5c8a41

**APÊNDICE**

**CÓDIGO PARA REALIZAR O DOWNLOAD DAS RPI’s**

# Montando o Drive

from google.colab import drive

drive.mount('/content/drive')

#MAPEAR PARA O LOCAL NO DRIVE ONDE os arquivos serão armazenados

if (True):

  import os

  os.chdir("drive/MyDrive/TCC")

# Instalação das bibliotecas

!pip install selenium

!apt-get update # to update ubuntu to correctly run apt install

!apt install chromium-chromedriver

!wget https://chromedriver.storage.googleapis.com/2.42/chromedriver\_linux64.zip  && unzip chromedriver\_linux64

import sys

sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')

from selenium import webdriver

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys

chrome\_options = webdriver.ChromeOptions()

chrome\_options.add\_argument('--headless')

chrome\_options.add\_argument('--no-sandbox')

chrome\_options.add\_argument('--disable-dev-shm-usage')

chrome\_options.add\_argument('--start-maximized')

chrome\_options.add\_argument('user-agent=Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36')

chrome\_options.add\_argument('--ignore-certificate-errors')

wd = webdriver.Chrome('chromedriver',chrome\_options=chrome\_options, service\_args=['--verbose', '--log-path=/tmp/chromedriver.log'])

from bs4 import BeautifulSoup

import time

import subprocess

import pandas as pd

# Download das RPI's

import time

# Acessando a URL

wd.get("http://revistas.inpi.gov.br/rpi/")

time.sleep(2) # Para cada ação, vamos inserir um tempo de espera (time.sleep()) no  # programa para a ação poder ser realizada:

# 1º ação do usuário na página, clicar no campo "Buscar Patentes"

busca\_patente = wd.find\_element\_by\_xpath('//a[@id="6"]')

busca\_patente.click()

time.sleep(2)

# 2º ação do usuário na página, clicar no tipo de pesquisa "por data"

tipo\_pesquisa = wd.find\_element\_by\_xpath('//input[@value="2"]')

tipo\_pesquisa.click()

time.sleep(2)

# 3º ação do usuário na página, entrar com a data inicial de publicação da rpi

data\_inicial = wd.find\_element\_by\_xpath('//input[@id="dataInicial"]')

data\_inicial.click()

print("Digite a data inicial no formato xx/xx/xxxx e tecle enter")

inicial=input() # utilizando o método input para entrar com a data inicial

data\_inicial.send\_keys(inicial)

time.sleep(2)

# 4º ação do usuário na página, entrar com a data final de publicação da rpi

data\_final = wd.find\_element\_by\_xpath('//input[@id="dataFinal"]')

data\_final.click()

print("Digite a data final no formato xx/xx/xxxx, e tecle enter")

final=input() # utilizando o método input para entrar com a data final

data\_final.send\_keys(final)

time.sleep(2)

# 5º ação do usuário, clicar no botão Buscar

button = wd.find\_elements\_by\_xpath('//button[@class="btn rst"]')[1]

button.click()

time.sleep(4)

# E, finalmente, extraindo o conteúdo da busca que aparece no site da rpi

result = wd.find\_element\_by\_xpath('//tbody[@id="resultado"]')

result\_rpi=result.get\_attribute('innerHTML')

time.sleep(3)

# Utilizando o beautifulsoup para traduzir a linguagem html

rpi\_page\_soup = BeautifulSoup(result\_rpi,'html.parser')

time.sleep(3)

# Pegando todos os arquivos

page\_links\_url = [a['href'] for a in rpi\_page\_soup.find\_all('a', href=True)]

time.sleep(3)

# Pegando os arquivos somente com extensão .zip

page\_zip\_urls = [zip\_url for zip\_url in page\_links\_url if ".zip" in zip\_url]

time.sleep(3)

# Importando as revistas

from datetime import datetime

start\_time = datetime.now()

for url in page\_zip\_urls:

  print("Extraindo o pdf da url: ", url)

  getRpm = 'wget --no-check-certificate -P ./ %s' %url

  subprocess.run(getRpm, shell=True)

end\_time = datetime.now()

print('\n')

print('Tempo total de importação das revistas: {}'.format(end\_time - start\_time))

print("Processo finalizado!")

# Como não faremos mais atividades de scrapping, vamos excluir os arquivos

# chromedriver, chromedriver\_linux64.zip, bem como o arquivo repetido P2130.zip.1

import os

os.remove('chromedriver')

os.remove('chromedriver\_linux64.zip')

os.remove('P2130.zip.1')

import os

os.chdir('../')

# Função para "unzip" os arquivos zipados

import zipfile

def un\_zipFiles(path):

    files=os.listdir(path)

    for file in files:

        if file.endswith('.zip'):

            filePath=path+'/'+file

            zip\_file = zipfile.ZipFile(filePath)

            for names in zip\_file.namelist():

                zip\_file.extract(names,path)

            zip\_file.close()

# Chamando a função:

path="TCC"

un\_zipFiles(path)

# Deletando os arquivos .zip e .xml

os.chdir('../')

dir\_name = "TCC"

test = os.listdir(dir\_name)

for item in test:

    if item.endswith(".zip") or item.endswith(".xml"):

        os.remove(os.path.join(dir\_name, item))

# Contando o número de arquivos txt

import glob

j=0

for x in glob.glob('\*'):

   j=j+1

j

**FUNÇÃO PARA ABRIR OS ARQUIVOS TXT E TRANFORMÁ-LOS EM UMA LISTA**

# Vamos definir uma função para abrir os arquivos txt retornando o mesmo transformado em uma lista:

# Nos testes acima, verificou-se que alguns arquivos txt abrem com o encoding utf-8 e outros com o iso-8859-1

def txtplista(t):

 try:

   with open(t, encoding = "utf-8") as f:

     list1 = [line.rstrip('\n') for line in f]

 except:

   with open(t, encoding = "ISO-8859-1") as f:

     list1 = [line.rstrip('\n') for line in f]

 f.close()

 return list1

**CÓDIGO PARA AUXILIAR NA IDENTIFICAÇÃO DE PADRÕES**

import glob

l=1

codigos=[]

rpi1=[]

rpi2=[]

for x in glob.glob("\*"):

  if x.endswith(".txt") or x.endswith(".TXT"):

    lista=txtplista(x)

    rpi1.append(lista[0])

    for i in lista:

      if (i[0:1]=='(' and i[3:4]==')') and (i[1:3].isnumeric()==True or i[1:3].isalpha()==True):

        n=i[0:6]

        codigos.append(n)

        rpi2.append(lista[0])

    print('rpi:',lista[0]," arquivo:",l)

    l=l+1

print('\n')

print("Processo de verificação Finalizado!")

f1=pd.DataFrame(codigos,columns=['codigos'])

f2=pd.DataFrame(rpi2,columns=['rpi'])

frames=[f2,f1]

cod\_rpi=pd.concat(frames, axis=1)

editado=cod\_rpi[['codigos', 'rpi']].groupby('codigos').agg(lambda x: list(set(x))).reset\_index()

print(editado.to\_string())

**FUNÇÃO remove\_(list)**

# Função para remover espaços no início e fim de cada linha da string

def remove\_(list1):

  i=0

  ni=0

  list2=[]

  for i in list1:

    i = i.lstrip()

    ni = i.rstrip()

    list2.append(ni)

  return list2

**FUNÇÃO c\_alta(list)**

# Função para transformar todas as primeiras 4 strings de cada linha da lista em caixa alta

def c\_alta(list1):

  i=0

  ni=0

  list2=[]

  for i in list1:

    if i[0:1]=='(' and i[1:3].isalpha()==True:

      ni=i.upper()

    else:

      ni=i

    list2.append(ni)

  return list2

**FUNÇÃO pedidos(list) PARA EXTRAIR AS INFORMAÇÕES DAS RPI’s**

# função para extrair as informações dasRPI’s e transformar estar informações em 3 dataframes

def pedidos(list1):

  # Criando listas vazias

  data\_rpi=[]

  despacho1=[]

  idpedido =[]

  data=[]

  original=[]

  ipc=[]

  titulo=[]

  depositante=[]

  data\_rpi2=[]

  despacho2=[]

  idpedido2=[]

  depositante2=[]

  data2=[]

  data\_rpi3=[]

  despacho3=[]

  idpedido3=[]

  novo\_numero=[]

  ndata\_rpi=''

  # vamos trabalhar apenas com a data de publicação da rpi

  # a linha abaixo retorna apenas os últimos caracteres da linha 0 da lista

  ndata\_rpi=list1[0][list1[0].rindex(' ')+1:]

  ndata\_rpi2=''

  ndata\_rpi2=ndata\_rpi

  ndata\_rpi3=''

  ndata\_rpi3=ndata\_rpi

  i=0  # linha da lista que será extraída a informação

  be=0

  ce=0

  de=0

  # Extração propriamente dita

  for j in list1:

      b=['(CD) 1.3','(CD) 3.1','(CD) 3.2','(CD) 2.4','(CD) 23.1.1']

      if j in b:

        if list1[i+1][5:7]=='PI' or list1[i+1][5:7]=='MU' or list1[i+1][5:6]=='C' or list1[i+1][5:7]=='BR':

          be+=1 # contagem dos pedidos com despachos contidos em b

          ndespacho1=''

          npedido =''

          ndata=''

          noriginal=''

          nipc=''

          ntitulo=''

          ndepositante=''

          ndespacho1=list1[i][5:]

          # Este loop dentro do despacho é para verificar se o valor correto está sendo extraído

          k=i+1

          while list1[k][0:5] != '(CD) ':

             if list1[k][0:5]=='(21) ' and list1[k][5:6].isupper()==True:

               npedido =list1[k][5:]

             elif list1[k][0:5]=='(22) ' and list1[k][5:6].isnumeric()==True:

               ndata=list1[k][5:]

             elif list1[k][0:5]=='(62) ' and list1[k][5:6].isupper()==True:

               noriginal=list1[k][5:]

             elif list1[k][0:5]=='(51) ' and list1[k][5:6].isupper()==True:

               nipc=list1[k][5:]

             elif list1[k][0:5]=='(54) ' and (list1[k][5:6].isalpha()==False or list1[k][5:6].isupper()==True):

               ntitulo=list1[k][5:]

             elif list1[k][0:5]=='(71) ':

               ndepositante=list1[k][5:]

             k+=1

          # append as listas

          data\_rpi.append(ndata\_rpi)

          idpedido.append(npedido)

          data.append(ndata)

          original.append(noriginal)

          ipc.append(nipc)

          titulo.append(ntitulo)

          depositante.append(ndepositante)

          despacho1.append(ndespacho1)

      c=['(CD) 6.1','(CD) 6.20','(CD) 6.21','(CD) 6.22','(CD) 7.1','(CD) 8.6','(CD) 8.11','(CD) 8.12','(CD) 9.1','(CD) 9.2','(CD) 11.1.1','(CD) 11.2', '(CD) 11.4','(CD) 11.6','(CD) 11.11','(CD) 11.18','(CD) 11.30','(CD) 11.31']

      if j in c:

        if list1[i+1][5:7]=='PI' or list1[i+1][5:7]=='MU' or list1[i+1][5:6]=='C' or list1[i+1][5:7]=='BR':

          ce+=1 # contagem dos pedidos com despachos contidos em c

          ndespacho2=''

          npedido2 =''

          ndata2=''

          ndepositante2=''

          ndespacho2=list1[i][5:]

          # Este loop dentro do despacho é para verificar se o valor correto está sendo extraído

          k=i+1

          while list1[k][0:5] != '(CD) ':

             if list1[k][0:5]=='(21) ' and list1[k][5:6].isupper()==True:

               npedido2 =list1[k][5:]

             elif list1[k][0:5]=='(22) ' and list1[k][5:6].isnumeric()==True:

               ndata2=list1[k][5:]

             elif list1[k][0:5]=='(71) ':

               ndepositante2=list1[k][5:]

             k+=1

          # append as listas

          data\_rpi2.append(ndata\_rpi2)

          idpedido2.append(npedido2)

          depositante2.append(ndepositante2)

          data2.append(ndata2)

          despacho2.append(ndespacho2)

      d=['(CD) 15.10','(CD) 15.12','(CD) 51']

      if j in d:

        if list1[i+1][5:7]=='PI' or list1[i+1][5:7]=='MU' or list1[i+1][5:6]=='C' or list1[i+1][5:7]=='BR':

          de+=1 # contagem dos pedidos com despachos contidos em d

          ndespacho3=''

          npedido3 =''

          nnovo\_numero=''

          # Este loop dentro do despacho é para verificar se o valor correto está sendo extraído

          k=i+1

          while list1[k][0:5] != '(CD) ':

             if list1[k][0:5]=='(21) ' and list1[k][5:6].isupper()==True:

               npedido3 =list1[k][5:]

             elif list1[k][0:5]=='(CO) ':

               nnovo\_numero=list1[k][5:]

             k+=1

          # append as listas

          data\_rpi3.append(ndata\_rpi3)

          idpedido3.append(npedido3)

          novo\_numero.append(nnovo\_numero)

      i+=1

  # Transformando as listas em dataframe

  pedido=pd.DataFrame(idpedido,columns=['idpedido'])

  data=pd.DataFrame(data,columns=['data\_deposito'])

  original=pd.DataFrame(original,columns=['original'])

  ipc=pd.DataFrame(ipc,columns=['ipc'])

  titulo=pd.DataFrame(titulo,columns=['titulo'])

  depositante=pd.DataFrame(depositante,columns=['depositante'])

  despacho=pd.DataFrame(despacho1,columns=['despacho'])

  data\_rpi=pd.DataFrame(data\_rpi,columns=['data\_rpi'])

  pedido2=pd.DataFrame(idpedido2,columns=['idpedido'])

  depositante2=pd.DataFrame(depositante2,columns=['depositante'])

  data2=pd.DataFrame(data2,columns=['data\_deposito'])

  despacho2=pd.DataFrame(despacho2,columns=['despacho'])

  data\_rpi2=pd.DataFrame(data\_rpi2,columns=['data\_rpi'])

  pedido3=pd.DataFrame(idpedido3,columns=['idpedido'])

  novo\_numero=pd.DataFrame(novo\_numero,columns=['novo\_numero'])

  data\_rpi3=pd.DataFrame(data\_rpi3,columns=['data\_rpi'])

  # Juntando os dataframes

  frames=[pedido,data,original,ipc,titulo,depositante,despacho,data\_rpi]

  pedido=pd.concat(frames, axis=1)

  frames2=[pedido2,data2,depositante2,despacho2,data\_rpi2]

  outros=pd.concat(frames2, axis=1)

  frames3=[pedido3,novo\_numero,data\_rpi3]

  nova\_num=pd.concat(frames3, axis=1)

  return pedido,outros,nova\_num,be,ce,de

**CÓDIGO DESENVOLVIDO PARA ABRIR CADA RPI E EXTRAIR AS INFORMAÇÕES DE INTERESSE**

import pandas as pd

import glob

from datetime import datetime

ped\_pub=[]

outros=[]

nova\_num=[]

xl=[]

al=[]

a1=0

b1=0

c1=0

start\_time = datetime.now()

l=1

for x in glob.glob("\*"):

  if x.endswith(".txt") or x.endswith(".TXT"):

    lista1=[]

    # Abrindo o arquivo e transformando-o em uma lista

    lista1=txtplista(x)

    # Tirando os espaços do início e do fim de cada linha da lista

    lista1=remove\_(lista1)

    # Passando todos os registros (cd) ou (co) para caixa alta

    lista1=c\_alta(lista1)

    # Extraindo as informações e salvando-as em 2 dataframes

    df1,df2,df3,a,b,c=pedidos(lista1)

    # append os df

    ped\_pub.append(df1)

    outros.append(df2)

    nova\_num.append(df3)

    xl.append(x)

    al.append(a)

    a1=a1+a

    b1=b1+b

    c1=c1+c

    print('Extraindo os dados do arquivo',x,'... ',' Nº de 1.3, 3.1, 3.2, 2.4 e 23.1.1:',a,' Arquivo nº:',l)

    l+=1

end\_time = datetime.now()

print('\n')

print('Total de linhas do dataframe com publicações 1.3, 3.1, 3.2, 2.4 e 23.1.1: ', a1)

print('\n')

print('Total de linhas do dataframe com outras publicações: ', b1)

print('\n')

print('Total de linhas do dataframe com pedidos renumerados: ', c1)

print('\n')

print('Tempo de extração das informações: {}'.format(end\_time - start\_time))

# Concatenando os df

print('\n')

print('Concatenando os dataframes...')

xdf = pd.DataFrame({'rpi':xl})

adf = pd.DataFrame({'qtd\_pub':al})

fr=[xdf,adf]

xdfadf=pd.concat(fr, axis=1)

dep\_pub\_total=pd.concat(ped\_pub,ignore\_index=True)

outros\_total=pd.concat(outros,ignore\_index=True)

nova\_num\_total=pd.concat(nova\_num,ignore\_index=True)

# Salvando os dataframes no formato csv

print('\n')

print('Salvando os dataframes no formato csv...')

xdfadf.to\_csv("xdfadf.csv", index=False)

dep\_pub\_total.to\_csv("dep\_pub\_total.csv", index=False)

outros\_total.to\_csv("outros\_total.csv", index=False)

nova\_num\_total.to\_csv("nova\_num\_total.csv", index=False)

print('\n')

print('Documentos salvos na pasta!')

print('\n')

print("Processo de extração Finalizado!")

**CÓDIGO DE EXTRAÇÃO DE INFORMAÇÕES DAS RPI’s ADAPTADO PARA EXTRAÇÃO DE DADOS DE RPI’s NO FORMATO PDF**

# Instalando a biblioteca Apache Tika

!pip install tika

from tika import parser

# Extraindo o texto do pdf...

dados\_pdf1 = parser.from\_file('PATENTES2118.pdf')

arquivo\_txt1 = dados\_pdf1['content']

dados\_pdf2 = parser.from\_file('PATENTES2119.pdf')

arquivo\_txt2 = dados\_pdf2['content']

# Formatando as linhas

textlines1 = [textline1 for textline1 in arquivo\_txt1.splitlines() if textline1 != '' and textline1 != ' ']

textlines2 = [textline2 for textline2 in arquivo\_txt2.splitlines() if textline2 != '' and textline2 != ' ']

# Tirando os espaços do início e do fim de cada linha da lista

l2118=remove\_(textlines1)

l2119=remove\_(textlines2)

# função para extrair as informações das RPI’s 2118 e 2119

def restante(list1,data\_rpi1):

  # Criando listas vazias

  data\_rpi=[]

  despacho1=[]

  idpedido =[]

  data=[]

  original=[]

  ipc=[]

  titulo=[]

  depositante=[]

  ndata\_rpi=''

  i=0  # linha da lista que será extraída a informação

  be=0

  k=0

  # Extração propriamente dita

  for j in list1:

        if list1[i][0:4]=='(21)' and k<(len(list1)-1) and (list1[i][list1[i].rindex(' ')+1:]=='1.3' or list1[i][list1[i].rindex(' ')+1:]=='3.1' or list1[i][list1[i].rindex(' ')+1:]=='3.2' or list1[i][list1[i].rindex(' ')+1:]=='2.4'):

          be+=1 # contagem dos pedidos com despachos contidos em b

          ndespacho1=''

          npedido =''

          ndata=''

          noriginal=''

          nipc=''

          ntitulo=''

          ndepositante=''

          ndata\_rpi=data\_rpi1

          npedido =j[5:17]

          ndata=j[26:37]

          ndespacho1=list1[i][list1[i].rindex(' ')+1:]

          # Este loop dentro do despacho é para verificar se o valor correto está sendo extraído

          k=i+1

          while list1[k][0:4] != '(21)':

             if list1[k][0:4]=='(62)':

               noriginal=list1[k][5:]

             elif list1[k][0:4]=='(51)':

               nipc=list1[k][5:]

             elif list1[k][0:4]=='(54)':

               ntitulo=list1[k][5:]

             elif list1[k][0:4]=='(71)':

               ndepositante=list1[k][5:]

             k+=1

          # append as listas

          data\_rpi.append(ndata\_rpi)

          idpedido.append(npedido)

          data.append(ndata)

          original.append(noriginal)

          ipc.append(nipc)

          titulo.append(ntitulo)

          depositante.append(ndepositante)

          despacho1.append(ndespacho1)

        i+=1

  # Transformando as listas em dataframe

  pedido=pd.DataFrame(idpedido,columns=['idpedido'])

  data=pd.DataFrame(data,columns=['data\_deposito'])

  original=pd.DataFrame(original,columns=['original'])

  ipc=pd.DataFrame(ipc,columns=['ipc'])

  titulo=pd.DataFrame(titulo,columns=['titulo'])

  depositante=pd.DataFrame(depositante,columns=['depositante'])

  despacho=pd.DataFrame(despacho1,columns=['despacho'])

  data\_rpi=pd.DataFrame(data\_rpi,columns=['data\_rpi'])

  # Juntando os dataframes

  frames=[pedido,data,original,ipc,titulo,depositante,despacho,data\_rpi]

  pedido=pd.concat(frames, axis=1)

  return pedido,be

# Extraindo as informações e salvando-as em 2 dataframes

data\_rpi1='09/08/2011'

data\_rpi2='16/08/2011'

df2118,a=restante(l2118,data\_rpi1)

df2119,a=restante(l2119,data\_rpi2)

**ETAPA DE TRANSFORMAÇÃO DOS DADOS EXTRAÍDOS DAS RPI’s**

import pandas as pd

# Lendo os arquivos que serão trabalhados

ped\_pub=pd.read\_csv('dep\_pub\_total.csv')

dfrestante=pd.read\_csv('dfrestante.csv')

outros=pd.read\_csv("outros\_total.csv")

nova\_num\_total\_ed=pd.read\_csv("nova\_num\_total\_ed.csv",encoding='iso-8859-1',delimiter=';')

# Passando as colunas despacho para o tipo string

ped\_pub['despacho'] = ped\_pub['despacho'].astype(str)

dfrestante['despacho']=dfrestante['despacho'].astype(str)

# Verificando o tipo de cada coluna n

print(ped\_pub.info())

print(outros.info())

print(nova\_num\_total\_ed.info())

print(dfrestante.info())

Criando uma nova coluna chave\_idpedido nos dataframes:

O dataframe nova\_num\_total\_ed foi previamente trado no excel, vamos apenas renomear as colunas:

nova\_num\_total\_ed.columns = ['chave\_idpedido', 'nova\_chave\_idpedido']

nova\_num\_total\_ed

ped\_pub['chave\_idpedido'] = ped\_pub['idpedido']

outros['chave\_idpedido'] = outros['idpedido']

dfrestante['chave\_idpedido']=dfrestante['idpedido']

# Tirando os espaços no início e no fim de cada string da coluna chave\_idpedido

ped\_pub['chave\_idpedido'] = ped\_pub['chave\_idpedido'].str.lstrip()

ped\_pub['chave\_idpedido'] = ped\_pub['chave\_idpedido'].str.rstrip()

outros['chave\_idpedido'] = outros['chave\_idpedido'].str.lstrip()

outros['chave\_idpedido'] = outros['chave\_idpedido'].str.rstrip()

dfrestante['chave\_idpedido']=dfrestante['chave\_idpedido'].str.lstrip()

dfrestante['chave\_idpedido']=dfrestante['chave\_idpedido'].str.rstrip()

# Excluindo os kind codes da coluna chave\_idpedido (consultar o site https://www.wipo.int/export/sites/www/standards/en/pdf/07-03-02.pdf para maiores informações)

x=['A2','A8','B1','B8','C8','E2','E8','F1','F8','G8','U2','U8','Y1','Y8','Z8']

for i in x:

  ped\_pub['chave\_idpedido'] = ped\_pub['chave\_idpedido'].map(lambda x: x.replace(i,''))

  outros['chave\_idpedido'] = outros['chave\_idpedido'].map(lambda x: x.replace(i,''))

  nova\_num\_total\_ed['chave\_idpedido'] = nova\_num\_total\_ed['chave\_idpedido'].map(lambda x: x.replace(i,''))

  nova\_num\_total\_ed['nova\_chave\_idpedido'] = nova\_num\_total\_ed['nova\_chave\_idpedido'].map(lambda x: x.replace(i,''))

# Tirando os espaços no início e no fim de cada string da coluna chave\_idpedido novamente

ped\_pub['chave\_idpedido'] = ped\_pub['chave\_idpedido'].str.lstrip()

ped\_pub['chave\_idpedido'] = ped\_pub['chave\_idpedido'].str.rstrip()

outros['chave\_idpedido'] = outros['chave\_idpedido'].str.lstrip()

outros['chave\_idpedido'] = outros['chave\_idpedido'].str.rstrip()

# Cada pedido de patente possui, antes do kind code, possuir um ifem seguindo de um número, que pode ser de 0 a 9. Vamos excluir estes números e o ifem também da chave\_idpedido

y=['-0','-1','-2','-3','-4','-5','-6','-7','-8','-9']

for j in y:

  ped\_pub['chave\_idpedido'] = ped\_pub['chave\_idpedido'].map(lambda x: x.replace(j,''))

  outros['chave\_idpedido'] = outros['chave\_idpedido'].map(lambda x: x.replace(j,''))

  dfrestante['chave\_idpedido']=dfrestante['chave\_idpedido'].map(lambda x: x.replace(j,''))

# Removendo os espaços em todas as strings da coluna chave\_idpedido

ped\_pub['chave\_idpedido'] = ped\_pub['chave\_idpedido'].str.replace(' ', '')

outros['chave\_idpedido'] = outros['chave\_idpedido'].str.replace(' ', '')

nova\_num\_total\_ed['chave\_idpedido'] = nova\_num\_total\_ed['chave\_idpedido'].str.replace(' ', '')

nova\_num\_total\_ed['nova\_chave\_idpedido'] = nova\_num\_total\_ed['nova\_chave\_idpedido'].str.replace(' ', '')

dfrestante['chave\_idpedido']=dfrestante['chave\_idpedido'].str.replace(' ', '')

# Renomeando as colunas

ped\_pub=ped\_pub[['chave\_idpedido','idpedido','data\_deposito', 'original','ipc','titulo','depositante','despacho','data\_rpi']]

outros=outros[['chave\_idpedido','idpedido','data\_deposito','depositante','despacho','data\_rpi']]

dfrestante=dfrestante[['chave\_idpedido','idpedido','data\_deposito', 'original','ipc','titulo','depositante','despacho','data\_rpi']]

Adicionando uma nova coluna ao dataframe contendo apenas a primeira classificação:

# Função para pegar apenas a primeira classificação da coluna ipc

def tira(test\_str):

    saida = ''

    v1 = 0

    v2 = 0

    for i in str(test\_str):

        if i == ',':

            v1 += 1

        elif i == '(':

            v2 += 1

        elif i == ',' and v1 > 0:

            v1 -= 1

        elif i == ')'and v2 > 0:

            v2 -= 1

        elif v1 == 0 and v2 == 0:

            saida += i

    return saida

# Extraindo somente a primeira classificação da coluna ipc, para criar a coluna ipc1

ped\_pub['ipc1'] = ped\_pub['ipc'].apply(lambda x: tira(x))

dfrestante['ipc1']=dfrestante['ipc'].apply(lambda x: tira(x))

Removendo os caracteres especiais da coluna titulo e aplicando a função upper em toda a coluna:

import re

ped\_pub['titulo1'] = ped\_pub['titulo'].map(lambda x: re.sub(r'\W+', ' ', str(x)))

ped\_pub['titulo1']=ped\_pub['titulo1'].str.upper()

dfrestante['titulo1'] = dfrestante['titulo'].map(lambda x: re.sub(r'\W+', ' ', str(x)))

dfrestante['titulo1']=dfrestante['titulo1'].str.upper()

Modificando a função que foi aplicada às classificações, vamos criar uma coluna somente com o nome do 1º depositante e tudo em maiúsculo:

# Função para pegar apenas o primeiro depositante com o país de origem

def tira1(test\_str):

    saida = ''

    v1 = 0

    for i in str(test\_str):

        if i == ';':

            v1 += 1

        elif i == ';'  and v1>0:

            v1 -= 1

        elif v1 == 0:

            saida += i

    return saida

# Extraindo somente o primeiro depositante com o país de origem

ped\_pub['atalho'] = ped\_pub['depositante'].apply(lambda x: tira1(x))

ped\_pub['atalho']=ped\_pub['atalho'].str.upper()

outros['atalho'] = outros['depositante'].apply(lambda x: tira1(x))

outros['atalho']=outros['atalho'].str.upper()

dfrestante['atalho'] = dfrestante['depositante'].apply(lambda x: tira1(x))

dfrestante['atalho']=dfrestante['atalho'].str.upper()

Vamos pegar somente o depositante da coluna atalho e criar uma coluna depositante1 para armazenar estes dados:

# Função para pegar apenas o primeiro depositante sem o país de origem

def tira2(test\_str):

    saida = ''

    v2 = 0

    for i in str(test\_str):

        if i == '(':

            v2 += 1

        elif i == ')'and v2 > 0:

            v2 -= 1

        elif v2 == 0:

            saida += i

    return saida

# Extraindo somente o primeiro depositante sem o país de origem

ped\_pub['depositante1'] = ped\_pub['atalho'].apply(lambda x: tira2(x))

outros['depositante1'] = outros['atalho'].apply(lambda x: tira2(x))

dfrestante['depositante1'] = dfrestante['atalho'].apply(lambda x: tira2(x))

Na coluna atalho podem exister alguns registros contendo outras strings entre parenteses, o que causa erro na extração do país do depositante. Vamos pegar então somente as 7 últimas posições da coluna atalho, para evitar que strings indesejadas sejam extraídas:

ped\_pub['atalho2'] = ped\_pub['atalho'].apply(lambda x: x[-7:])

outros['atalho2'] = outros['atalho'].apply(lambda x: x[-7:])

dfrestante['atalho2'] = dfrestante['atalho'].apply(lambda x: x[-7:])

# Função para pegar apenas o país do depositante

def tira3(test\_str):

    saida = ''

    v2 = 0

    for i in str(test\_str):

        if i == '(':

            v2 += 1

        elif i == ')'and v2 > 0:

            v2 -= 1

        elif v2 != 0:

            saida += i

    return saida

ped\_pub['atalho2'] = ped\_pub['atalho2'].apply(lambda x: tira3(x))

outros['atalho2'] = outros['atalho2'].apply(lambda x: tira3(x))

dfrestante['atalho2'] = dfrestante['atalho2'].apply(lambda x: tira3(x))

dfrestante = dfrestante.reset\_index(drop=True)

# Extraindo somente o primeiro país e o estado, caso seja depositante nacional

b=ped\_pub['atalho2']

origem\_depos=[]

origem\_depos\_estado=[]

for i in b:

  if len(i)==2:

    nb=i

    nb1=0

  else:

    nb=i[0:2]

    if i[0:2]=='BR':

      nb1=i[-2:]

    elif i[0:2]!='BR':

        nb1=0

  origem\_depos.append(nb)

  origem\_depos\_estado.append(nb1)

origem\_depos=pd.DataFrame(origem\_depos,columns=['ori\_depos'])

origem\_depos\_estado=pd.DataFrame(origem\_depos\_estado,columns=['ori\_depos\_est'])

ped\_pub['ori\_depos']=origem\_depos

frames=[ped\_pub,origem\_depos\_estado]

ped\_pub=pd.concat(frames, axis=1)

b=outros['atalho2']

origem\_depos=[]

origem\_depos\_estado=[]

for i in b:

  if len(i)==2:

    nb=i

    nb1=0

  else:

    nb=i[0:2]

    if i[0:2]=='BR':

      nb1=i[-2:]

    elif i[0:2]!='BR':

        nb1=0

  origem\_depos.append(nb)

  origem\_depos\_estado.append(nb1)

origem\_depos=pd.DataFrame(origem\_depos,columns=['ori\_depos'])

origem\_depos\_estado=pd.DataFrame(origem\_depos\_estado,columns=['ori\_depos\_est'])

outros['ori\_depos']=origem\_depos

frames=[outros,origem\_depos\_estado]

outros=pd.concat(frames, axis=1)

b=dfrestante['atalho2']

origem\_depos=[]

origem\_depos\_estado=[]

for i in b:

  if len(i)==2:

    nb=i

    nb1=0

  else:

    nb=i[0:2]

    if i[0:2]=='BR':

      nb1=i[-2:]

    elif i[0:2]!='BR':

      nb1=0

  origem\_depos.append(nb)

  origem\_depos\_estado.append(nb1)

origem\_depos=pd.DataFrame(origem\_depos,columns=['ori\_depos'])

origem\_depos\_estado=pd.DataFrame(origem\_depos\_estado,columns=['ori\_depos\_est'])

dfrestante['ori\_depos']=origem\_depos

frames=[dfrestante,origem\_depos\_estado]

dfrestante=pd.concat(frames, axis=1)

# Deletando as colunas atalho criadas

del ped\_pub['atalho']

del ped\_pub['atalho2']

del outros['atalho']

del outros['atalho2']

del dfrestante['atalho']

del dfrestante['atalho2']

Alterando as ordens da coluna e salvando em um novo dataframe:

ped\_pub\_final=ped\_pub[['chave\_idpedido','idpedido','data\_deposito','original','ipc1','titulo1','depositante1','ori\_depos','ori\_depos\_est','despacho','data\_rpi']]

outros\_final=outros[['chave\_idpedido','idpedido','data\_deposito','depositante1','ori\_depos','ori\_depos\_est','despacho','data\_rpi']]

dfrestante=dfrestante[['chave\_idpedido','idpedido','data\_deposito','original','ipc1','titulo1','depositante1','ori\_depos','ori\_depos\_est','despacho','data\_rpi']]

Renumerando o index do dataframe para ficar em ordem:

ped\_pub\_final = ped\_pub\_final.reset\_index(drop=True)

outros\_final = outros\_final.reset\_index(drop=True)

dfrestante = dfrestante.reset\_index(drop=True)

Vamos eliminar as barras de todas as datas para poder padronizar um formato único:

ped\_pub\_final['data\_deposito1'] = ped\_pub\_final['data\_deposito'].apply(lambda x: str(x).replace('/',''))

ped\_pub\_final['data\_rpi1'] = ped\_pub\_final['data\_rpi'].apply(lambda x: str(x).replace('/',''))

outros\_final['data\_deposito1'] = outros['data\_deposito'].apply(lambda x: str(x).replace('/',''))

outros\_final['data\_rpi1'] = outros['data\_rpi'].apply(lambda x: str(x).replace('/',''))

# Removendo os espaços das strings das colunas que contém data

dfrestante['data\_rpi'] = dfrestante['data\_rpi'].apply(lambda x: str(x).replace(' ',''))

dfrestante['data\_deposito']= dfrestante['data\_deposito'].apply(lambda x: str(x).replace(' ',''))

# Função para transformar a data no formato data do python:

from datetime import datetime

def data\_f(coluna):

  data=[]

  ndata=''

  lista=[]

  for i in coluna:

    if len(str(i))==8:

      y=str(i[-4:])

      m=str(i[2:4])

      d=str(i[0:2])

      z=y+'-'+m+'-'+d

    elif len(str(i))==6:

      y='19'+str(i[-2:])

      m=str(i[2:4])

      d=str(i[0:2])

      if m.isnumeric()==False:

        m='10'

      elif d.isnumeric()==False:

        d='10'

      z=y+'-'+m+'-'+d

    elif len(str(i))<6 :

      z='1991-01-01'

    lista.append(z)

  for i in lista:

    try:

      ndata = datetime.strptime(i, '%Y-%m-%d').date()

    except:

      ndata = datetime.strptime('1991-01-01', '%Y-%m-%d').date()

    data.append(ndata)

  data=pd.DataFrame(data,columns=['data\_'])

  return data

# Função para transformar a data no formato data do python:

from datetime import datetime

def data\_r(coluna):

  data=[]

  ndata=''

  lista=[]

  for i in coluna:

    if len(str(i))==10:

      y=str(i[-4:])

      m=str(i[3:5])

      d=str(i[0:2])

      z=y+'-'+m+'-'+d

    lista.append(z)

  for i in lista:

    ndata = datetime.strptime(i, '%Y-%m-%d').date()

    data.append(ndata)

  data=pd.DataFrame(data,columns=['data\_'])

  return data

# aplicando as funções

ped\_pub\_final['data\_deposito2']=data\_f(ped\_pub\_final['data\_deposito1'])

ped\_pub\_final['data\_rpi2']=data\_f(ped\_pub\_final['data\_rpi1'])

outros\_final['data\_deposito2']=data\_f(outros\_final['data\_deposito1'])

outros\_final['data\_rpi2']=data\_f(outros\_final['data\_rpi1'])

dfrestante['data\_deposito1']=data\_r(dfrestante['data\_deposito'])

dfrestante['data\_rpi1']=data\_r(dfrestante['data\_rpi'])

# Reordenando as colunas

ped\_pub\_final=ped\_pub\_final[['chave\_idpedido','idpedido','data\_deposito2','original','ipc1','titulo1','depositante1','ori\_depos','ori\_depos\_est','despacho','data\_rpi2']]

dfrestante=dfrestante[['chave\_idpedido','idpedido','data\_deposito1','original','ipc1','titulo1','depositante1','ori\_depos','ori\_depos\_est','despacho','data\_rpi1']]

outros\_f=outros\_final[['chave\_idpedido','idpedido','data\_deposito2','depositante1','ori\_depos','ori\_depos\_est','despacho','data\_rpi2']]

outros\_f['despacho'].unique()

outros\_f.columns = ['chave\_idpedido','idpedido','data\_deposito','depositante','ori\_depos','ori\_depos\_est','despacho','data\_rpi']

np.size(outros\_f['depositante'].unique())

np.size(outros\_f['ori\_depos'].unique())

# Função para tirar as vírgulas dos depositantes

def tira4(test\_str):

    saida = ''

    v1 = 0

    for i in str(test\_str):

        if i == ',':

            v1 += 1

        elif i == ','  and v1>0:

            v1 -= 1

        elif v1 == 0:

            saida += i

    return saida

outros\_f['depositante2'] = outros\_f['depositante'].apply(lambda x: tira4(x))

outros\_f

Removendo os caracteres especiais dos depositantes:

import re

outros\_f['depositante2'] = outros\_f['depositante2'].map(lambda x: re.sub(r'\W+', ' ', str(x)))

outros\_f['depositante2']=outros\_f['depositante2'].str.upper()

outros\_f

Removendo a acentuação dos depositantes, para evitar diferenças:

outros\_f['depositante2']=outros\_f['depositante2'].str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8')

outros\_f

del outros\_f['depositante']

outros\_f=outros\_f[['chave\_idpedido','idpedido','data\_deposito','depositante2','ori\_depos','ori\_depos\_est','despacho','data\_rpi']]

outros\_f.columns = ['chave\_idpedido','idpedido','data\_deposito','depositante','ori\_depos','ori\_depos\_est','despacho','data\_rpi']

np.size(outros\_f['depositante'].unique())

# salvando como arquivo csv para trabalhar no arquivo posteriormente:

outros\_f.to\_csv("outros\_f.csv", index=False)

Voltando para o dataframe ped\_pub\_final e dfrestante:

ped\_pub\_final.sort\_values(by=['data\_deposito2']).head(60)

ped\_pub\_final.sort\_values(by=['data\_deposito2']).tail(60)

Vamos separar as classificações em três níveis. O primeiro nível, é só a primeira letra da classificação i[0:1], o segundo nível trata dos 3 primeiros digitos da classificação i[0:3] e o terceiro nível de detalhe são os 4 primeiros dígitos i[0:4]. Lembrando que cada nível apresenta um detalhamento maior da tecnologia a que se refere aquela classificação:

ped\_pub\_final['ipc1\_1'] = ped\_pub\_final['ipc1'].apply(lambda x: x[0:1])

dfrestante['ipc1\_1'] = dfrestante['ipc1'].apply(lambda x: x[0:1])

ped\_pub\_final['ipc1\_2'] = ped\_pub\_final['ipc1'].apply(lambda x: x[0:3])

dfrestante['ipc1\_2'] = dfrestante['ipc1'].apply(lambda x: x[0:3])

ped\_pub\_final['ipc1\_3'] = ped\_pub\_final['ipc1'].apply(lambda x: x[0:4])

dfrestante['ipc1\_3'] = dfrestante['ipc1'].apply(lambda x: x[0:4])

Removendo algumas repetições que ocorrem em algumas linhas da coluna depositante1:

def tira4(test\_str):

    saida = ''

    v1 = 0

    for i in str(test\_str):

        if i == ',':

            v1 += 1

        elif i == ','  and v1>0:

            v1 -= 1

        elif v1 == 0:

            saida += i

    return saida

ped\_pub\_final['depositante2'] = ped\_pub\_final['depositante1'].apply(lambda x: tira4(x))

dfrestante['depositante2'] = dfrestante['depositante1'].apply(lambda x: tira4(x))

import re

ped\_pub\_final['depositante3'] = ped\_pub\_final['depositante2'].map(lambda x: re.sub(r'\W+', ' ', str(x)))

ped\_pub\_final['depositante3']=ped\_pub\_final['depositante3'].str.upper()

dfrestante['depositante3'] = dfrestante['depositante2'].map(lambda x: re.sub(r'\W+', ' ', str(x)))

dfrestante['depositante3']=dfrestante['depositante3'].str.upper()

ped\_pub\_final['depositante4']=ped\_pub\_final['depositante3'].str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8')

dfrestante['depositante4']=dfrestante['depositante3'].str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8')

ped\_pub\_final

ped\_pub\_final['titulo2']=ped\_pub\_final['titulo1'].str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8')

dfrestante['titulo2']=dfrestante['titulo1'].str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8')

ped\_pub\_f=ped\_pub\_final[['chave\_idpedido','idpedido','data\_deposito2','original','ipc1','ipc1\_1','ipc1\_2','ipc1\_3','titulo2','depositante4','ori\_depos','ori\_depos\_est','despacho','data\_rpi2']]

ped\_pub\_f.columns = ['chave\_idpedido','idpedido','data\_deposito','original','ipc','ipc\_1','ipc\_2','ipc\_3','titulo','depositante','ori\_depos','ori\_depos\_est','despacho','data\_rpi']

dfrestante=dfrestante[['chave\_idpedido','idpedido','data\_deposito1','original','ipc1','ipc1\_1','ipc1\_2','ipc1\_3','titulo2','depositante4','ori\_depos','ori\_depos\_est','despacho','data\_rpi1']]

dfrestante.columns = ['chave\_idpedido','idpedido','data\_deposito','original','ipc','ipc\_1','ipc\_2','ipc\_3','titulo','depositante','ori\_depos','ori\_depos\_est','despacho','data\_rpi']

Verificando se há idpedidos repetidos:

dupl=ped\_pub\_f['chave\_idpedido'].value\_counts().rename\_axis('unique\_values').to\_frame('counts')

dupl2=dfrestante['chave\_idpedido'].value\_counts().rename\_axis('unique\_values').to\_frame('counts')

dupl.head(60)

dupl2.head(60)

print(ped\_pub\_f['ori\_depos'].value\_counts().rename\_axis('unique\_values').to\_frame('counts').to\_string())

ped\_pub\_f['depositante'].value\_counts().rename\_axis('unique\_values').to\_frame('counts').head(60)

ped\_pub\_f['depositante'].value\_counts().rename\_axis('unique\_values').to\_frame('counts').tail(60)

ped\_pub\_f.loc[ped\_pub\_f['chave\_idpedido'] == 'BR112014000556']

ped\_pub\_f.loc[ped\_pub\_f['chave\_idpedido'] == 'BR112014025730']

ped\_pub\_f.loc[ped\_pub\_f['chave\_idpedido'] == 'BR112015014844']

Como observado acima, podem haver simples repetições da publicação do pedido por causa da anulação da publicação anterior ou qualquer outro motivo para correção de outras informações. Assim, manteremos o registro com a última publicação na rpi, removendo os registros anteriores.

ped\_pub\_f=ped\_pub\_f.sort\_values(by=['chave\_idpedido','data\_rpi'])

ped\_pub\_f = ped\_pub\_f.reset\_index(drop=True)

ped\_pub\_f.loc[ped\_pub\_f['chave\_idpedido'] == 'BR102017020904']

ped\_pub\_f.loc[ped\_pub\_f['chave\_idpedido'] == 'BR112014000976']

ped\_pub\_f.loc[ped\_pub\_f['chave\_idpedido'] == 'BR112015014844']

Eliminando, então, as linhas com idpedido repetidas, mantendo o último idpedido (data mais recente da rpi):

ped\_pub\_f=ped\_pub\_f.drop\_duplicates(subset='chave\_idpedido', keep="last")

ped\_pub\_f['chave\_idpedido'].value\_counts().rename\_axis('unique\_values').to\_frame('counts')

Agora todas as idpedido repetidas foram removidas, mantendo-se apenas as que foram publicadas na rpi mais recente!

ped\_pub\_f=ped\_pub\_f.sort\_values(by=['data\_deposito'])

ped\_pub\_f = ped\_pub\_f.reset\_index(drop=True)

ped\_pub\_f.to\_csv("ped\_pub\_f.csv", index=False)

dfrestante.to\_csv('dfrestante\_f.csv',index=False)

Vamos criar novas colunas descrevendo as tecnologias a que cada ipc representa de acordo com as definições da OMPI.

ped\_pub\_f=pd.read\_csv("ped\_pub\_f.csv")

dfrestante\_f=pd.read\_csv('dfrestante\_f.csv')

area\_campo\_ompi=pd.read\_csv('area\_campo\_ompi.csv',encoding='iso-8859-1',delimiter=';')

ped\_pub\_f=ped\_pub\_f.sort\_values(by=['ipc'])

ped\_pub\_f = ped\_pub\_f.reset\_index(drop=True)

dfrestante\_f=dfrestante\_f.sort\_values(by=['ipc'])

verif=ped\_pub\_f.loc[ped\_pub\_f['ipc'].isna() == True].sort\_values(by=['data\_deposito'])

verif

verif.loc[verif['data\_deposito'] < '1993-01-01']

verif.loc[verif['data\_deposito'] > '2018-12-31']

Removendo os espaços entre as classificações:

ped\_pub\_f['ipc'] = ped\_pub\_f['ipc'].str.replace(' ', '')

dfrestante\_f['ipc'] = dfrestante\_f['ipc'].str.replace(' ', '')

Inserindo uma nova coluna no df para utilizar as definições tecnologicas de arcodo com a OMPI:

at=[]

n=''

for i in ped\_pub\_f['ipc']:

  i=str(i)

  if i[0:4]=='A61K':

    if i[0:6]=='A61K8/':

      n=i[0:6]

    else:

      n=i[0:4]

  elif i[0:4]=='B01D':

    if i[0:7]=='B01D45/' or i[0:7]=='B01D46/' or i[0:7]=='B01D47/' or i[0:7]=='B01D49/' or i[0:7]=='B01D50/' or i[0:7]=='B01D51/' or i[0:7]=='B01D52/':

      n=i[0:7]

    else:

        n=i[0:4]

  elif i[0:4]=='C13B':

    if i[0:6]=='C13B5/':

      n=i[0:6]

    elif i[0:7]=='C13B15/'  or i[0:7]=='C13B25/'  or i[0:7]=='C13B45/':

      n=i[0:7]

    else:

      n=i[0:4]

  elif i[0:4]=='E01F':

    if i[0:6]=='E01F8/':

      n=i[0:6]

    else:

      n=i[0:4]

  elif i[0:4]=='G01N':

    if i[0:7]=='G01N33/':

      n=i[0:7]

    else:

      n=i[0:4]

  elif i[0:4]=='H04N':

    if i[0:6]=='H04N1/':

      n=i[0:6]

    elif i[0:7]=='H04N21/':

      n=i[0:7]

    else:

      n=i[0:4]

  else:

    n=i[0:4]

  at.append(n)

ipc\_at=pd.DataFrame(at,columns=['ipc\_at'])

ped\_pub\_f['ipc\_at']=ipc\_at

at=[]

n=''

for i in dfrestante\_f['ipc']:

  i=str(i)

  if i[0:4]=='A61K':

    if i[0:6]=='A61K8/':

      n=i[0:6]

    else:

      n=i[0:4]

  elif i[0:4]=='B01D':

    if i[0:7]=='B01D45/' or i[0:7]=='B01D46/' or i[0:7]=='B01D47/' or i[0:7]=='B01D49/' or i[0:7]=='B01D50/' or i[0:7]=='B01D51/' or i[0:7]=='B01D52/':

      n=i[0:7]

    else:

        n=i[0:4]

  elif i[0:4]=='C13B':

    if i[0:6]=='C13B5/':

      n=i[0:6]

    elif i[0:7]=='C13B15/'  or i[0:7]=='C13B25/'  or i[0:7]=='C13B45/':

      n=i[0:7]

    else:

      n=i[0:4]

  elif i[0:4]=='E01F':

    if i[0:6]=='E01F8/':

      n=i[0:6]

    else:

      n=i[0:4]

  elif i[0:4]=='G01N':

    if i[0:7]=='G01N33/':

      n=i[0:7]

    else:

      n=i[0:4]

  elif i[0:4]=='H04N':

    if i[0:6]=='H04N1/':

      n=i[0:6]

    elif i[0:7]=='H04N21/':

      n=i[0:7]

    else:

      n=i[0:4]

  else:

    n=i[0:4]

  at.append(n)

ipc\_at=pd.DataFrame(at,columns=['ipc\_at'])

dfrestante\_f['ipc\_at']=ipc\_at

Aplicando o método merge nos dois df, teremos uma nova coluna informando a área eo campo tecnológico de acordo com a tabela area\_campo\_ompi:

result = pd.merge(ped\_pub\_f,area\_campo\_ompi, how='outer')

rdfrestante\_f = pd.merge(dfrestante\_f,area\_campo\_ompi, how='outer')

Vamos guardar as linhas com NaN do df acima, talvez possa ser útil mais adiante. Depois vamos deletar estas linhas.

nan\_result=result[619157:]

nan\_result

nan\_result.to\_csv("nan\_result.csv", index=False)

result = result.drop(result[619157:].index)

result.tail(60)

rdfrestante\_f = rdfrestante\_f.drop(rdfrestante\_f[979:].index)

rdfrestante\_f.tail(60)

Podemos criar mais uma coluna, informando se o depositante é de origem internacional (Não-Residente) ou nacional (Residente):

def nac\_int(x):

  if x=='BR':

    y='residente'

  else:

    y='nao\_residente'

  return (y)

result['Nac\_Inter']=result['ori\_depos'].apply(lambda x: nac\_int(x))

rdfrestante\_f['Nac\_Inter']=rdfrestante\_f['ori\_depos'].apply(lambda x: nac\_int(x))

result

result['Nac\_Inter'].value\_counts().rename\_axis('unique\_values').to\_frame('counts')

result['area\_campo\_ompi'].value\_counts().rename\_axis('unique\_values').to\_frame('counts')

result=result.sort\_values(by=['data\_deposito'])

result = result.reset\_index(drop=True)

result.loc[result['data\_deposito'] < '1993-01-01']

result = result.reset\_index(drop=True)

result

result.loc[result['data\_deposito'] >'2020-12-31']

result.to\_csv("result\_f.csv", index=False)

rdfrestante\_f.to\_csv("rdfrestante\_f\_f.csv", index=False)

outros\_f['despacho'].value\_counts().rename\_axis('unique\_values').to\_frame('counts')

Cada despacho significa:

11.1.1 - Arquivamento - Art. 33 Parágravo único da LPI

11.11 - Arquivamento - Art.17§ 2º da LPI

11.18 - Arquivamento definitivo por não anuência relacionada com o Art. 229-C da LPI.

11.2 - Arquivamento definitivo - Art. 36 §1° da LPI

11.4 - Arquivamento - Art.38 § 2º da LPI

11.5 - Arquivamento - Art.34 da LPI

11.6 - Arquivamento do Pedido - Art.216 § 2º da LPI

11.30 - Arquivamento Definitivo - Art. 18 § 1º da Lei 5772/71

11.31 - Arquivamento Definitivo - Falta de Cumprimento de exigência

11.6 - Arquivamento do Pedido - Art.216 § 2º da LPI

6.1 - Exigência - Art.36 da LPI

6.20 - Exigência Pré-Exame - Art. 34 da LPI

6.21 - Exigência preliminar - pedidos com buscas realizadas por outros Escritórios de Patentes

6.22 - Exigência preliminar - pedidos sem buscas realizadas por outros Escritórios de Patentes

7.1 - Conhecimento do Parecer Técnico

8.6 - Arquivamento - Art.86 da LPI

8.11 - Manutenção do Arquivamento

8.12 - Arquivamento Definitivo

9.1 - Deferimento

9.2 – Indeferimento

outros\_f.to\_csv("outros\_f.csv", index=False)

# Removendo os espaços das colunas

nova\_num\_total\_ed.columns = ['chave\_original', 'chave\_idpedido']

nova\_num\_total\_ed['chave\_original'] = nova\_num\_total\_ed['chave\_original'].str.replace(' ', '')

nova\_num\_total\_ed['chave\_idpedido'] = nova\_num\_total\_ed['chave\_idpedido'].str.replace(' ', '')

Após as transformações acima, juntamos os pedidos restantes das RPI's 2118 e 2119 ao dataframe com as outras publicações. As verificações de duplicidade e verificação dos pedidos com renumeração foram feitas no excel, pelo método procv.

Primeiro, antes de fazer o merge, tivemos que preparar os arquivos para fazer o merge de forma correta. Dividimos o arquivo com os pedidos publicados em 2. Um, em que não houve renumeração (result\_f\_p1xm, para fazer um merge apenas) e outro (result\_f\_p2xm), contendo uma coluna a mais com a nova numeração do pedido. No arquivo em que temos somente pedidos que foram renumerados, teremos que fazer 2x o merge. No primeiro merge, faremos considerando a numeração antiga, na segunda, a nova numeração, e depois verificaremos via excel. Os arquivos já preparados, serão mostrados abaixo:

O arquivo que faremos o merge é o outros\_f\_pm, também preparado no excel, em que algumas verificações de data e informações foram feitas, bem como a exclusão dos pedidos com data de depósito inferior a 1993.

outros\_f\_pm=pd.read\_csv('outros\_f\_pm.csv',encoding='iso-8859-1',delimiter=';')

result\_f\_p1xm=pd.read\_csv('result\_f\_p1xm.csv',encoding='iso-8859-1',delimiter=';')

result\_f\_p2xm=pd.read\_csv('result\_f\_p2xm.csv',encoding='iso-8859-1',delimiter=';')

Vamos juntar os dois dataframes utilizando o método merge:

def e\_desp(df):

  df1=[]

  c=['6.1','6.20','6.21','6.22','7.1','8.6','8.11','8.12','9.1','9.2','11.1.1','11.2','11.4','11.6','11.11','11.18','11.30','11.31']

  for desp in c:

    at=df.loc[df['despacho'] == desp].sort\_values(by=['chave\_idpedido','data\_rpi'])

    at = at.reset\_index(drop=True)

    at['chave\_idpedido'].value\_counts()

    at[desp] = at.groupby('chave\_idpedido')['chave\_idpedido'].transform('count')

    at=at.drop\_duplicates(subset='chave\_idpedido', keep="last")

    del at['despacho']

    at=at[['chave\_idpedido',desp,'data\_rpi']]

    at = at.reset\_index(drop=True)

    df1.append(at)

  return df1[0], df1[1], df1[2], df1[3], df1[4], df1[5], df1[6], df1[7], df1[8], df1[9], df1[10], df1[11], df1[12], df1[13], df1[14], df1[15], df1[16], df1[17]

df\_6\_1, df\_6\_20, df\_6\_21, df\_6\_22, df\_7\_1, df\_8\_6, df\_8\_11, df\_8\_12, df\_9\_1, df\_9\_2, df\_11\_1\_1, df\_11\_2,df\_11\_4,df\_11\_6,df\_11\_11,df\_11\_18,df\_11\_30,df\_11\_31=e\_desp(outros\_f\_pm)

a=[df\_6\_1,df\_6\_20,df\_6\_21,df\_6\_22,df\_7\_1,df\_8\_6,df\_8\_11,df\_8\_12,df\_9\_1,df\_9\_2,df\_11\_1\_1,df\_11\_2,df\_11\_4,df\_11\_6,df\_11\_11,df\_11\_18,df\_11\_30,df\_11\_31]

a

for i in a:

  result\_f\_p1xm=pd.merge(result\_f\_p1xm,i,how='outer',on=['chave\_idpedido'])

  result\_f\_p2xm=pd.merge(result\_f\_p2xm,i,how='outer',on=['chave\_idpedido']) # primeiro merge, a chave idpedido do dataframe result\_f\_p2xm está ativada na 1º coluna

result\_f\_p2xm = result\_f\_p2xm.drop(result\_f\_p2xm[1303:].index)

Alterando a chave do result\_f\_p2xm para a nova numeração, pela simples alteração do nome da coluna e refazendo o merge:

result\_f\_p2xm.rename(columns={'chave\_idpedido': 'num\_anterior', 'nova\_num': 'chave\_idpedido'}, inplace=True)

for i in a:

  result\_f\_p2xm=pd.merge(result\_f\_p2xm,i,how='outer',on=['chave\_idpedido']) # segundo merge, a chave idpedido do dataframe result\_f\_p2xm está ativada na coluna com a nova numeração

result\_f\_p2xm = result\_f\_p2xm.drop(result\_f\_p2xm[1303:].index)

result\_f\_p2xm.rename(columns={'num\_anterior':'chave\_idpedido', 'chave\_idpedido':'nova\_num'}, inplace=True)

result\_f\_p2xm

Este dataframe completo, contendo os dados bibliográficos dos pedidos e as ações que estes pedidos sofreram ao longo do tempo não terá utilidade neste trabalho específico, mas somente para uso interno do INPI.

Os dados utilizados serão os dados compilados dos dataframes que contém somente os dados bibliográficos e do dataframe outros.

dupl=result\_f\_p1xm['chave\_idpedido'].value\_counts().rename\_axis('unique\_values').to\_frame('counts')

dupl

dupl=result\_f\_p2xm['chave\_idpedido'].value\_counts().rename\_axis('unique\_values').to\_frame('counts')

dupl

result\_f\_p1xm.to\_csv("result\_f\_1.csv", index=False)

result\_f\_p2xm.to\_csv("result\_f\_2.csv", index=False)

Terminamos a segunda etapa! Vamos à terceira etapa do trabalho.

from google.colab import files

files.download("result\_f\_1.csv")

files.download("result\_f\_2.csv")

from google.colab import files

files.download("outros\_f.csv")

**CÓDIGO BASE UTILIZADO PARA GERAR GRÁFICOS COM ANIMAÇÃO (BAR CHART RACE)**

# Importando bibliotecas para gráfico de barras horizontal

import matplotlib.ticker as ticker

import matplotlib.animation as animation

from IPython.display import HTML

# Lendo o arquivo despachoxano.csv

pais\_ano=pd.read\_csv('paisxano.csv',encoding='iso-8859-1',delimiter=';')

pais\_ano

# substituindo os valores nan por 0 no dataframe

pais\_ano=pais\_ano.fillna(0)

pais\_ano

# Pegando apenas o ano de 2016

current\_Ano = 2016

pais\_ano\_1 = (pais\_ano[pais\_ano['Ano'].eq(current\_Ano)]

       .sort\_values(by='Depósitos', ascending=False).head(10))

pais\_ano\_1

# Vamos adicionar cores de acordo com o continente de cada país

colors = dict(zip(

    ['Oceania', 'Europe', 'Asia', 'Latin America', 'North America'],

    ['#adb0ff', '#aafbff', '#90d595', '#e48381', '#f7bb5f']

))

group\_lk = pais\_ano.set\_index('País')['Continente'].to\_dict()

fig, ax = plt.subplots(figsize=(15, 8))

pais\_ano\_1 = pais\_ano\_1[::-1]   # flip values from top to bottom

# pass colors values to `color=`

ax.barh(pais\_ano\_1['País'], pais\_ano\_1['Depósitos'], color=[colors[group\_lk[x]] for x in pais\_ano\_1['País']])

# iterate over the values to plot labels and values

for i, (Depósitos, País) in enumerate(zip(pais\_ano\_1['Depósitos'], pais\_ano\_1['País'])):

    ax.text(Depósitos, i,     País,            ha='right')  # País: name

    ax.text(Depósitos, i-.25, group\_lk[País],  ha='right')  # Asia: Continente

    ax.text(Depósitos, i,     Depósitos,           ha='left')   # Depósitos

# Add year right middle portion of canvas

ax.text(1, 0.4, current\_Ano, transform=ax.transAxes, size=46, ha='right')

# Melhorando a apresentação do gráfico

fig, ax = plt.subplots(figsize=(15, 8))

def draw\_barchart(ano):

    pais\_ano\_1 = pais\_ano[pais\_ano['Ano'].eq(ano)].sort\_values(by='Depósitos', ascending=True).tail(10)

    ax.clear()

    ax.barh(pais\_ano\_1['País'], pais\_ano\_1['Depósitos'], color=[colors[group\_lk[x]] for x in pais\_ano\_1['País']])

    dx = pais\_ano\_1['Depósitos'].max() / 200

    for i, (Depósitos, País) in enumerate(zip(pais\_ano\_1['Depósitos'], pais\_ano\_1['País'])):

        ax.text(Depósitos-dx, i,     País,           size=14, weight=600, ha='right', va='bottom')

        ax.text(Depósitos-dx, i-.25, group\_lk[País], size=10, color='#444444', ha='right', va='baseline')

        ax.text(Depósitos+dx, i,     f'{Depósitos:,.0f}',  size=14, ha='left',  va='center')

    # ... polished styles

    ax.text(1, 0.4, ano, transform=ax.transAxes, color='#777777', size=46, ha='right', weight=800)

    ax.xaxis.set\_major\_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

    ax.xaxis.set\_ticks\_position('top')

    ax.tick\_params(axis='x', colors='#777777', labelsize=12)

    ax.set\_yticks([])

    ax.margins(0, 0.01)

    ax.grid(which='major', axis='x', linestyle='-')

    ax.set\_axisbelow(True)

    ax.text(0, 1.12, 'Países com maior número de depósitos de pedidos de patente no Brasil',

            transform=ax.transAxes, size=24, weight=600, ha='left')

    plt.box(False)

draw\_barchart(2016)

# Finalmente, utilizando o recurso de animação do matplotlib

import matplotlib.animation as animation

from IPython.display import HTML

fig, ax = plt.subplots(figsize=(18, 10))

animator = animation.FuncAnimation(fig, draw\_barchart, frames=range(1993, 2017))

HTML(animator.to\_jshtml())

# or use animator.to\_html5\_video() or animator.save()

**CÓDIGO BASE PARA GERAR A REDE NEURAL, TREINAR E TESTAR A REDE**

Parte 1 – Pre-processamento

# Importando bibliotecas

import numpy

import matplotlib.pyplot as plt

import pandas

import math

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM, Conv1D, Dropout

from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

numpy.random.seed(7)

# Carrega apenas a coluna com o total de doentes por mês

dataframe=pandas.read\_csv('pub\_mes.csv',usecols=[1], engine='python',skipfooter=3,encoding='iso-8859-1',delimiter=';')

dataframe.head(3)

dataframe.plot()

Aqui tratamos os dados que vamos usar. Primeiro ajustamos a escala dos dados para ficarem entre 0 e 1. A seguir, dividimos os dados entre treino (67% primeiros meses)e testes (33% dos meses finais da série):

#Converte a coluna do dataframe pandas em um vetor numpy

dataset = dataframe.values

dataset = dataset.astype('float32')

look\_back = 12

# Divide os dados de treino (2/3) e teste (1/3)

# Note que a divisão não é aleatória, mas sim sequencial

train\_size = int(len(dataset) \* 0.67)

test\_size = len(dataset) - train\_size

train, test = dataset[0:train\_size,:], dataset[train\_size-look\_back-1:len(dataset),:]

Aqui criamos nossos pares de dados $X$ e $Y$, onde $Y*\_{t} = X\_*{t-1}$ (para look\_bak=1). Uma outra forma de pensar no valor de $Y$ para um dado $X$ é que ele é o próximo $X$ na série temporal.

# Recebe uma série e converte em uma matriz com séries deslocadas.

def create\_dataset(dataset, look\_back=1, std=1):

    dataX, dataY = [], []

    for i in range(len(dataset)-look\_back):

        a = dataset[i:(i+look\_back), 0]-dataset[i, 0]

        a /= std

        dataX.append(a)

        dataY.append(dataset[i + look\_back, 0]-dataset[i + look\_back-1, 0])

    return numpy.array(dataX), numpy.array(dataY)

# reshape into X=t and Y=t+1

std = train[:, 0].std()

trainX, trainY = create\_dataset(train, look\_back, std)

testX, testY = create\_dataset(test, look\_back, std)

# shape is [samples, time steps, features]

trainX = trainX.reshape(-1, look\_back, 1)

testX = testX.reshape(-1, look\_back, 1)

trainY = trainY / 30

testY = testY / 30

trainX.shape, testX.shape

Parte 2 – Construção da RNN

Agora criamos nossa rede LSTM com 32 recorrências (usa até 20 pontos da série para tentar prever o próximo número) e a treinamos.

model = Sequential()

model.add(LSTM(20, input\_shape=(look\_back, 1), return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.summary()

callbacks = [

    ReduceLROnPlateau(patience=10, factor=0.5, verbose=True),

    ModelCheckpoint('best.model.h5', save\_best\_only=True),

    EarlyStopping(patience=25, verbose=True)

]

history = model.fit(trainX, trainY, epochs=5000, batch\_size=24, validation\_data=(testX, testY),

                    verbose=0, callbacks=callbacks)

df\_history = pandas.DataFrame(history.history)

ax = df\_history[['val\_loss', 'loss']].plot(figsize=(10, 5))

df\_history['lr'].plot(ax=ax.twinx(), color='gray')

Parte 3 – Previsões e análise dos resultados

# Realiza as previsões. Notar que a utilidade de prever trainX é nenhuma. Serve apenas para exibir no gráfico.

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

Imprime o gráfico da previsão (em vermelho)

# shift train predictions for plotting

trainPredictPlot = (trainPredict.ravel() \* 30) + dataset[look\_back:len(trainPredict)+look\_back, 0]

# shift test predictions for plotting

testPredictPlot = (testPredict.ravel() \* 30) + dataset[len(trainPredict)+(look\_back)-1:len(dataset), 0]

# plot baseline and predictions

plt.figure(figsize=(20, 10))

plt.plot(dataset)

plt.plot(look\_back+numpy.arange(len(trainPredictPlot)), trainPredictPlot)

plt.plot(look\_back+numpy.arange(len(testPredictPlot))+len(trainPredictPlot)-1, testPredictPlot)

plt.show()

Parte 4- Métricas de avaliação

# Calcula os erros de previsão

trainScore = math.sqrt(mean\_squared\_error(trainY, trainPredict[:,0]))

print('Train Score: %.2f RMSE' % (trainScore))

testScore = math.sqrt(mean\_squared\_error(testY, testPredict[:,0]))

print('Test Score: %.2f RMSE' % (testScore))

**REDE NEURAL FINAL ADOTADA PARA PREVISÃO**

# Rede neuronal adotada, contendo camada única com 500 neurônios e otimizador igual a sgd

model = Sequential()

model.add(LSTM(500, input\_shape=(look\_back, 1), return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='sgd')

model.summary()

# Treinamento com 60 épocas

callbacks = [

    ReduceLROnPlateau(patience=10, factor=0.5, verbose=True),

    ModelCheckpoint('best.model.h5', save\_best\_only=True),

    EarlyStopping(patience=25, verbose=True)

]

history = model.fit(trainX, trainY, epochs=60, batch\_size=24, validation\_data=(testX, testY),

                    verbose=0, callbacks=callbacks)