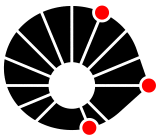

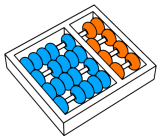

Pontuar Textos Utilizando RNNs

Gustavo G. Avena

Relatório Técnico - IC-PFG-18-32
Projeto Final de Graduação
2018 - Novembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Pontuar Textos Utilizando RNNs

Gustavo Galvão Avena

Resumo

Esse trabalho tem como objetivo inserir pontuação em textos em português produzidos por serviços de transcrições de voz. Para solucionar o problema, uma rede neural recorrente bi-direcional foi treinada e avaliada com múltiplos datasets distintos, construídos com dados de três fontes: legendas de vídeos, artigos de notícia e transcrições de discursos. As vantagens e desvantagens de cada um dos datasets foram avaliadas junto com os resultados de precisão e acurácia de cada um dos seus respectivos modelos.

Além disso, foi treinado um modelo "híbrido"juntando os discursos com artigos de tópicos selecionados. Esse modelo obteve a melhor performance, com um F1-Score de 74.9 no seu test dataset.

O tópico central do texto pontuado não teve uma influência grande no resultado de nenhum dos modelos, mas a fusão de artigos para o treino de um modelo híbrido levou a uma melhora significativa de performance.

Sumário

1 Introdução

Serviços de speech-to-text possuem inúmeras aplicações em setores de atendimento a clientes, análise de dados e comunicação. Porém, esses serviços fornecem transcrições sem pontuação, dificultando a leitura e a compreensão do texto. O objetivo desse trabalho é prover um sistema de pontuação que possa ser associado a esses serviços para, possivelmente, melhorar seus resultados. Os modelos treinados se restringiram a três sinais de pontuação: ponto, vírgula e interrogação.

Ao contrário das Redes Neurais Feedforward, Redes Neurais Recorrentes (RNNs) possuem "memória", porque podem utilizar seu estado interno para processar sequências de entradas. Por isto, as RNNs são bastante utilizadas para resolver problemas na área de linguística como tradução e reconhecimento de voz. Baseado no projeto Punctuator[1], que busca resolver o mesmo problema para textos em inglês e estoniano, foram utilizados modelos de redes neurais recorrentes utilizando a biblioteca Theano[5] para abordar o problema. Vários scripts foram implementados em Python para a obtenção e limpeza de dados, construção de datasets e configuração do projeto no Google Colab[4].

2 Redes Neurais

2.1 Redes Neurais Artificiais

Pesquisas relacionadas a representação de redes neurais através de modelos matemáticos e computacionais começaram há muitas décadas atrás. Esses modelos, chamados de Redes Neurais Artificiais (RNAs), buscam reproduzir componentes do sistema nervoso central de animais, como neurônios, para a execução de tarefas específicas após um processo de aprendizagem.

Existem vários tipos de RNAs, sendo o mais simples uma rede constituída de múltiplos *perceptrons*, unidades que buscam simular neurônios, conectados por arestas com pesos distintos. Essas redes podem possuir múltiplas camadas e a informação é sempre propagada em uma direção (da entrada para saída), por isso são conhecidas como redes *feedforward*. A figura 1 mostra um exemplo de uma rede neural feedforward.

Durante o treino das RNAs, o usuário fornece dados de entrada com suas respectivas saídas e a rede atualiza os pesos das arestas buscando definir um modelo cada vez mais preciso. Devido a sua complexidade, essas redes são capazes de representar hipóteses não-lineares. Porém, dependendo do tamanho da rede, o processo de aprendizagem pode ser bastante custoso, por se tratar de operações envolvendo matrizes de grande porte.

Atualmente, com o grande progresso das GPUs, o treino e uso de RNAs tem se tornado mais acessível, porque esses componentes são capazes de reduzir bastante o tempo necessário para executar esses cálculos.

2.2 Redes Neurais Recorrentes

Em Redes Neurais Recorrentes, a saída das unidades pode ser utilizada como entrada de alguma unidade da sua camada ou de camadas anteriores. Essa característica fornece um

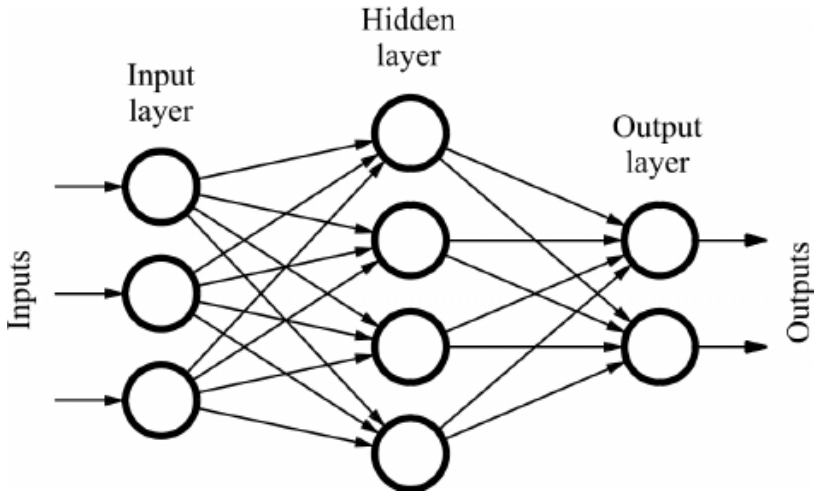

Figura 1: Rede Neural Artificial

tipo de "memória"a rede neural[7], já que a saída de cada célula depende das saídas de células anteriores e posteriores.

Redes neurais recorrentes são bastante utilizadas em problemas com dependência temporal ou sequencial, como reconhecimento de voz e tradução de máquina.

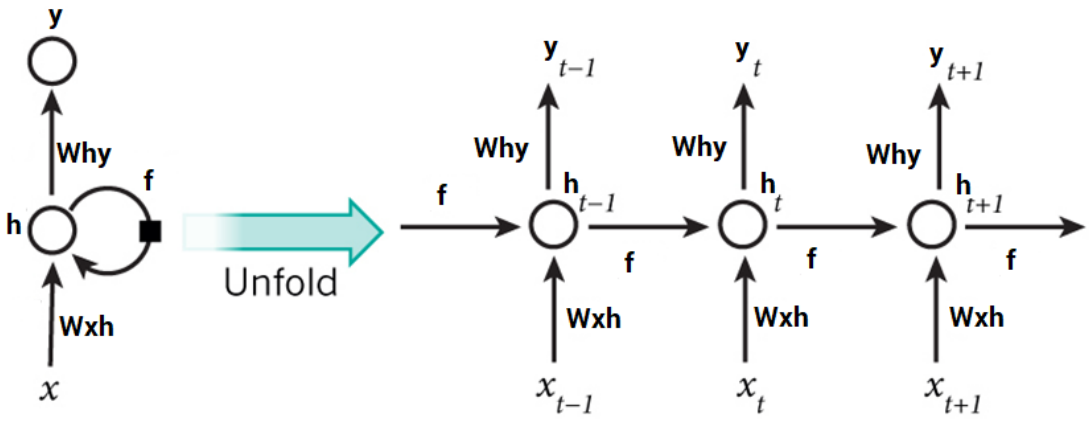

Figura 2: Rede Neural Recorrente

2.3 Gated Recurrent Unit (GRU)

Gated Recurrent Unit, ou GRU, é um tipo de unidade que pode ser usada para compor redes neurais recorrentes. Sua estrutura pode ser vista na figura 3. Ela foi introduzida em 2014 e tem uma estrutura semelhante a unidades LSTM (Long Short-Term Memory), mas demonstraram melhor performance em datasets pequenos[6].

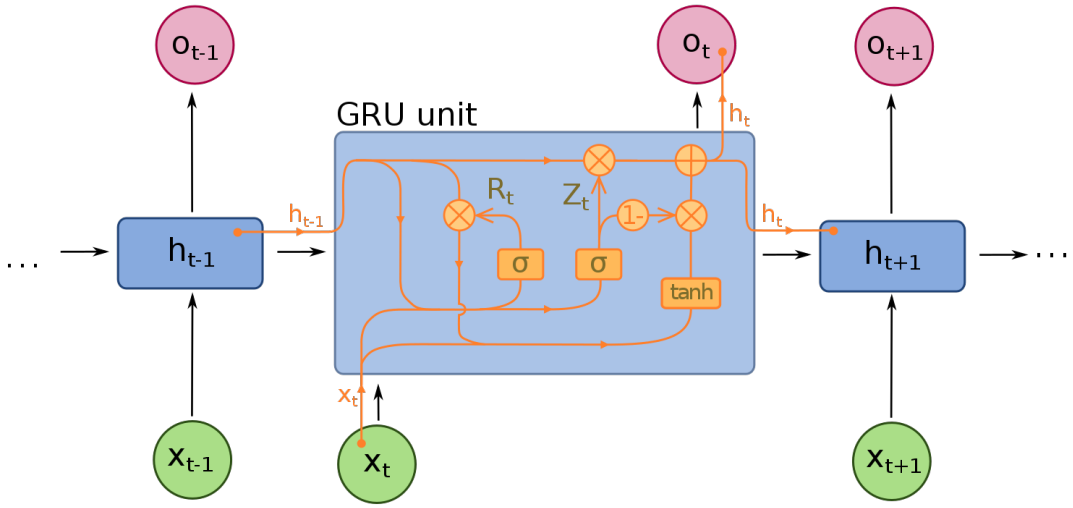

Figura 3: Gated Recurrent Unit

3 Punctuator

Na computação, a reutilização de código é algo bastante comum porque evita que muitas pessoas realizem trabalho repetido desnecessariamente, contribuindo para a criação de mais projetos com impactos maiores. Por isso, no começo desse trabalho, foram buscadas soluções para o problema em questão, para que não fosse necessário começar todo o trabalho do zero.

Nessa busca foi encontrado o *Punctuator*[1], um projeto que utiliza Redes Neurais Recorrentes Bi-Direcionais com mecanismos de atenção para pontuar textos em inglês e estoniano. Ele possui código que define o modelo da RNN, prepara os dados, treina o modelo e avalia sua performance. Tudo isso é feito em Python, utilizando a biblioteca Theano.

Há também a opção treinar modelos em dois estágios, levando em consideração pausas e aspectos prosódicos dos textos. Essa funcionalidade não foi utilizada nesse projeto por não ter sido encontrada uma fonte de dados que pudesse fornecer informações prosódicas de transcrições.

O uso do Punctuator nesse projeto permitiu que a maior parte do trabalho envolvesse a busca e preparação de dados de boa qualidade para treinar os modelos.

3.1 Theano

Theano é uma biblioteca em Python que otimiza o cálculo de expressões matemáticas, especialmente com matrizes, através de compilação. O uso de Python, uma linguagem de alto nível e muito poderosa, com várias otimizações que são compiladas em C, tornou Theano uma biblioteca muito atraente para projetos envolvendo aprendizado de máquina e redes neurais, já que esses projetos envolvem muitos cálculos matemáticos com matrizes.

Theano possui código aberto e o seu desenvolvimento foi encerrado em 2017 devido a presença de outras bibliotecas que possuíam as mesmas funcionalidade (e.g. TensorFlow, desenvolvida pelo Google).

4 Metodologia

4.1 Preparação de Datasets

Em qualquer projeto de aprendizado de máquina, a qualidade dos dados utilizados é essencial para obter-se bons resultados. Como as especificações da rede neural e o código usados nesse projeto já tinham sido desenvolvidos, a maior preocupação foi a obtenção e preparação dos dados que seriam utilizados na rede. O Punctuator obteve bons resultados com datasets de 40 milhões de palavras, então esse tamanho foi utilizado como meta nesse projeto.

Todos os datasets utilizados para treino, uso e avaliação do modelo tiveram sua pontuação "simplificada", buscando a otimização dos resultados. Eram considerados somente os sinais de ponto, vírgula e interrogação, então todos os outros sinais foram mapeados para um desses três. Esse mapeamento pode ser visto na tabela 1.

Sinal Original	Substituto
Exclamação (!)	Ponto (.)
Dois Pontos (:)	Vírgula (,)
Ponto e vírgula (;)	Ponto (.)
Hífen (-)	Vírgula (,)

Tabela 1: Mapeamento de sinais de pontuação

4.1.1 Limpeza

Todas as fontes de dados utilizadas forneciam vários arquivos de texto, que deviam ser agrupados para posteriormente ser dividido em três datasets: treino, validação e teste, cada um em seu arquivo de texto. Porém, antes de junta-los, cada um desses arquivos passava por uma limpeza, essencial para o processo de *feature engineering* que busca otimizar a qualidade do dataset final. Esse processo de limpeza era específico de cada fonte de dados, mas consistia basicamente na remoção de expressões repetidas e caracteres peculiares.

Praticamente todo essa limpeza foi executada utilizando expressões regulares. A biblioteca *re* de Python foi utilizada para encontrar strings com padrões definidos pelas expressões regulares e remove-las ou substitui-las por strings apropriadas. Antes de executar essa remoção, foi necessário definir para cada fonte quais eram os padrões que deveriam ser encontrados. Para isto, foi utilizado um trecho de código que fornece os n-grams mais frequentes de um texto. Esse código encontrou os n-grams frequentes de tamanho 1 a 5 e seu resultado foi utilizado para definir as expressões regulares utilizadas na limpeza de cada conjunto de dados. O processo de limpeza e preparação dos dados era executado por um script que percorria os arquivos em ordem aleatória, executava a limpeza e os adicionava a um arquivo final. Portanto, no fim desse processo, cada fonte de dados fornecia um arquivo de texto contendo todos os dados (e.g. *dataset_artigos_clean.txt*), que posteriormente passava pelo processo de split, como descrito na próxima seção.

4.1.2 Split

O processo de treino de uma rede neural não consiste simplesmente da inserção de dados de entrada. Os algoritmos de treino utilizam um training dataset em conjunto com um validation dataset para otimizar os parâmetros da rede. Após o processo de treino, a performance da rede deve ser avaliada utilizando-se o test dataset.

Seguindo a sugestão do Punctuator, os valores de 0.6, 0.2 e 0.2 foram escolhidos inicialmente como a fração do dataset completo utilizada para construir os datasets de treino, validação e teste, respectivamente. Posteriormente, os valores de 0.7, 0.15 e 0.15 foram utilizados para avaliar como o aumento do training dataset poderia melhorar a performance do modelo. Durante o relatório, o valor do split irá se referir ao fração do dataset completo utilizada no training dataset (e.g. 0.6 ou 0.7).

Como as features utilizadas no modelo eram extraídas de textos, foi extremamente importante definir como os dados serão divididos sem alterar sua qualidade ou inserir qualquer tipo de informação errada. Isso poderia acontecer se, por exemplo, frases fossem cortadas pela metade ou tivessem sua pontuação removida.

Algumas ideias para divisão foram avaliadas, como separar os datasets pelo número de arquivos (e.g. de 1000 arquivos, 600 vão para treino e 200 para validação e teste) ou pelo número de palavras. A primeira solução não levaria a um resultado muito consistente, porque o tamanho de cada arquivo variava bastante. No dataset de artigos, por exemplo, o tamanho dos arquivos variava de 50 bytes a 30 kilobytes.

A segunda solução, por outro lado, poderia afetar a qualidade dos dados, cortando frases pela metade e prejudicando o resultado do modelo. Para corrigir esse problema, o numero de palavras foi utilizado em conjunto com a pontuação do texto. O número de palavras do arquivo obtido após a limpeza era contado, e seu valor era utilizado em conjunto com os valores de split mencionados anteriormente. Porém, o script garantia que nenhuma frase seria cortada pela metade, levando em consideração sinais de pontuação que indicam fim de sentenças (e.g. ponto, interrogação).

4.2 Tamanho da Hidden Layer

Além de buscar uma melhora da performance variando o tamanho do traininig dataset, foram testados múltiplos valores de tamanho da hidden layer, n_{hid}, que representa o número de células presentes em cada hidden layer. Além de 256, como recomendando pelo Punctuator, os valores de 128, 384 e 512 foram testados. O valor de $n_{hid} = 512$ gerou erros durante o treino de múltiplos modelos, então foi descartado.

Os resultados melhoraram ao aumentar n_{hid} de 128 para 256, mas o aumento para 384 não ofereceu uma melhoria significativa ao levar em consideração o tempo adicional para treinar o modelo.

4.3 Fontes de Dados

As três fontes de dados utilizadas para criação de datasets e seus respectivos tamanhos podem ser vistas na tabela 2.

Fonte	Tamanho (em palavras)
Legendas de vídeos do YouTube	820 mil
Artigos de notícias do InfoMoney e globo.com	3.6 milhões
Transcrições de discursos do Senado Federal	46.6 milhões

Tabela 2: Fontes de dados utilizadas para criação de datasets e seus respectivos tamanhos

A partir desses dados, datasets foram criados para treinar modelos com hiperparâmetros diferentes que tiveram sua performance avaliada. Nas próximas três subseções, será detalhado o processo de obtenção e limpeza dos dados de cada uma dessas fontes.

4.3.1 Youtube

A utilização de captions de vídeos do YouTube foi a escolha inicial para treinar o modelo. A primeira tarefa foi buscar múltiplos canais brasileiros que produziam vídeos com captions de boa qualidade. Foram escolhidos canais como *TedX Brasil*, *Manual do Mundo* e *Você Sabia*. Após a seleção do vídeos, o programa *youtubedl*[3], que permite o download de recursos do YouTube através de uma CLI, foi utilizado para obter as legendas.

Os vídeos obtidos foram utilizados para formar um dataset completo de cerca de 820 mil palavras [1]. As vantagens do uso de legendas para treinar o modelo eram principalmente a diversidade das pessoas que falavam (i.e. idade, gênero, origem) e a variedade dos tópicos abordados, que evitava que somente um tipo de conteúdo fosse utilizado no treino.

A maior desvantagem desse dataset era a quantidade de informação inútil que teve que ser removida dos dados originais. Foram apagadas muitas frases repetidas, que eram extremamente comuns em certos canais devido a presença de vinhetas e "catch phrases" em todos os seus vídeos. Além disso, muitos vídeos eram compostos de diálogos, o que gerava textos bem diferentes do que era esperado para o modelo. Os diálogos continham muitas frases curtas, que não forneciam muito contexto para treinar o modelo.

4.3.2 Artigos

Buscando melhorar a performance e resolver os problemas apresentados pelo dataset anterior, foi utilizado um API de artigos de notícia chamado *NewsAPI*[2] para montar um novo dataset. Este API fornece URLs de artigos de vários tópicos da *globo.com* e da *InfoMoney*. Após a obtenção de muitos URLs, foi realizado scrapping e download dos textos utilizando a biblioteca de Python chamada *Newspaper*[8].

Esse dataset resolveu alguns problemas do anterior, como o volume de dados disponíveis e, além disso, facilitou a obtenção dos dados, feita através de um script simples.

Porém, os dados apresentavam outros problemas. Primeiro, poucos artigos possuíam algum tipo de transcrição, que era o tipo de texto que essa rede neural é treinada pra pontuar. Além disso, o problema das expressões repetidas ainda persistiu nesse dataset. A remoção dos n-grams consumiu um tempo significativo, porque novas expressões repetidas

[1] Dataset completo, antes do split

apareciam frequentemente, então era necessário remover essas expressões e recomeçar o processo de limpeza e treino.

4.3.3 Discursos

A terceira fonte de dados foi o Serviços de Dados Abertos do Senado Federal. Esse serviço possui uma vasta quantidade de transcrições de discursos, que são facilmente acessíveis através de um API. Além de resolver o problema do volume pequeno de dados, conseguindo atingir o objetivo de 40 milhões de palavras no dataset de treino[2], os discursos do senado não possuíam tanto conteúdo que precisava ser removido. Sua limpeza foi relativamente simples, feita através da remoção de nomes de senadores e alteração de alguns pronomes de tratamento e títulos que apareciam frequentemente.

Os tópicos abordados nos discursos abrangiam diferentes áreas, mas a maior parte era relacionada à política. Além disso, a linguagem dos discursos possui uma formalidade acima da média, por serem retirados do Senado. Essa formalidade maior do que os outros datasets diverge um pouco dos tipos de texto que a rede é projetada para pontuar, mas os benefícios obtidos pela organização e o quantidade dos dados compensaram esse defeito, o que levou a resultados melhores do que os datasets anteriores, como será discutido nos próximos capítulos.

4.4 Dataset Extra: Discursos + Artigos

Para avaliar a influência dos tópicos na performance dos modelos, foram feitos alguns testes utilizando datasets que continham artigos de tópicos específicos: política, esportes, tecnologia e famosos. Foi criado um arquivo para cada tópico e todos passaram pelo processo de clean up descrito na seção 4.1.1. Nas próximas seções, esses arquivos serão chamados de "arquivos de tópicos"ou "datasets de tópicos"(e.g. dataset de tecnologia).

Primeiro, o modelo de discursos com split de 0.7 e $n_{hid} = 256$ foi utilizado para pontuar cada um desses arquivos[3]. Depois, foi treinado um novo modelo, utilizando um dataset "híbrido"formado por todos os discursos e os datasets de tópicos. O dataset híbrido, que completo possuía 49.3 milhões de palavras, passou por um split de 0.7 e treinou um modelo com $n_{hid} = 256$. Esse novo modelo "híbrido"foi utilizado para pontuar os arquivos de tópicos, o test dataset híbrido e o test dataset do modelo de discursos original. Os resultados desse processo podem ser vistos na seção 5.4.

4.5 Google Colab

Como todo o projeto foi implementado no sistema operacional MacOS, não foi possível utilizar otimizações do Theano com o uso da GPU da máquina local. Portanto, uma tarefa importante do projeto consistiu na preparação de uma infraestrutura que permitisse o treino, uso e avaliação dos modelos em tempos aceitáveis. O uso do Google Colab, que fornece acesso a GPUs *Tesla K80* sem nenhum custo ao desenvolvedor, foi essencial para melhorar

[2]Dataset completo foi de 46.6 milhões de palavras

[3]A partir deste ponto, este modelo será chamado de "modelo de discursos original"

a infraestrutura do projeto. Algumas tarefas, como treino de modelos, tiveram uma redução de 95,8% no seu tempo de execução.

Para permitir a utilização do Colab, muita pesquisa foi feita sobre a biblioteca *libgpuarray* utilizada pelo Theano e todo o código do Punctuator foi convertido para Python 3.

5 Resultados

Os resultados dos modelos de cada fonte de dados, expressos em F1-Score, podem ser vistos na tabela 3.

O F1-Score é uma métrica estatística que leva em consideração *precision* e *recall*. Antes de explicar o que esse valores representam, é importante definir o que significa um positivo no nosso modelo. Um positivo será sempre referente a um sinal de pontuação específica, isto é, a presença ou não de um sinal de pontuação em um ponto do texto. Por exemplo, se temos no texto original a frase "Nossa, que legal."e o modelo produz "Nossa. Que legal.", temos um falso negativo de uma vírgula, um falso positivo de um ponto e um positivo verdadeiro de um ponto.

Precision representa o número de positivos verdadeiros classificados pelo modelo dividido pelo número total de positivos classificados pelo modelo (verdadeiros e falsos). Recall representa o número de positivos verdadeiros dividido pelo número total de positivos reais (valores que eram pra ter sido classificados como positivos, ou seja, positivos verdadeiros + negativos falsos). O F1-Score é obtido através de uma média ponderada dos valores de precision e recall de cada sinal de pontuação.

	128		256		384	
	0.6	0.7	0.6	0.7	0.6	0.7
YouTube	37.1	39.5	36.4	42.3	40.6	35.4
Artigos	57.4	58.9	58.4	58.7	59.1	56.4
Discursos	72.5	72.9	74.5	75.2	75.4	75.6

Tabela 3: F1-Score dos modelos de acordo com o split e n_{hid}

5.1 YouTube

O melhor modelo obtido a partir de legendas do YouTube foi o de split de 0.7 e $n_{hid} = 256$. Esse modelo obteve um F1-Score de 42.3, com precision igual a 42.0 e recall igual a 42.5.

5.2 Artigos

O uso de artigos para treinar o modelo levou a um F1-Score máximo de 59.1, no modelo com split 0.6 e $n_{hid} = 384$.

5.3 Discursos

As transcrições de discursos do senado geraram os melhores resultados. O modelo treinado com $n_{hid} = 384$ obteve um F1-Score de 75.6, um pouco maior do que o score de 75.2 obtido pelo modelo com $n_{hid} = 256$. Esses scores altos provavelmente são reflexo do tamanho do dataset utilizado no treino. Os discursos geraram um dataset completo com aproximadamente 46.6 milhões de palavras, mais de 10 vezes o tamanho do dataset de artigos, que consistia de aproximadamente 3.6 milhões.

5.4 Discursos + Artigos

Os resultados dos experimentos que avaliaram a influência dos tópicos dos textos nos resultados (seção 4.4) podem ser vistos na tabela 4 [4].

O uso do modelo de discursos original para pontuar os arquivos de tópicos gerou resultados relativamente bons (score médio de 55.5), considerando que esses textos foram obtidos de uma fonte completamente diferente. É importante observar que o arquivo de política não obteve o melhor resultado dos quatro, como era esperado, já que a maior parte dos discursos são relacionados a política. Isso mostra que o tópico abordado pelos textos não tem necessariamente uma influência muito grande na performance do modelo.

Foi muito interessante observar o impacto que o uso dos artigos teve sobre o novo modelo. A melhor performance em todos os arquivos foi a do modelo híbrido. Ele não só aumentou o score médio dos arquivos de tópicos para 73.7, como obteve um score de 78.0 no dataset de discursos original.

	Politica	Tecnologia	Famosos	Esportes	Discursos + Artigos (Test Dataset)	Discursos (Test Dataset)
Discursos	56.5	57.3	53.4	54.8	-	75.2
Discursos + Artigos	75.7	75.9	71.9	71.2	74.9	78.0

Tabela 4: F1-Score do modelo de discursos original e do novo modelo criado com discursos e artigos

6 Conclusões

As transcrições dos discursos do senado, apesar de não formarem um dataset livre de defeitos, foram essenciais para obter obter boas performances nos modelos devido a qualidade e a grande quantidade de textos disponíveis para treino.

Após alguns testes com o modelo de discursos original, foi observado que o tópico central do texto não necessariamente tem uma grande influência na performance do modelo. Mas foi possível obter resultados significativamente melhores após a junção de artigos com o dataset de discursos para a criação de um modelo "híbrido".

Para buscar uma melhora ainda maior, próximos passos que poderiam ser tomados são: treinar novos modelos com mais fontes de dados diferentes juntas, avaliar se o aumento dos

[4] O modelo original de discursos não foi utilizado para pontuar o test dataset do novo modelo.

datasets pode contribuir para a performance e analisar outros possíveis valores de split e n_{hid}.

Referências

[1] Ottokar Tilk e Tanel Alumäe. "Bidirectional Recurrent Neural Network with Attention Mechanism for Punctuation Restoration". Em: *Interspeech 2016*. 2016.

[2] News API. *News API Documentation*. URL: `https://newsapi.org/docs`.

[3] Ricardo Garcia Gonzalez. *youtube-dl*. URL: `https://github.com/rg3/youtube-dl`.

[4] Google. *Introduction to Google Colab*.

[5] LISA lab. *Theano - API Documentation*. URL: `http://deeplearning.net/software/theano/library/index.html`.

[6] Wikipedia. *Gated Recurrent Unit*. URL: `https://en.wikipedia.org/wiki/Gated_recurrent_unit`.

[7] Wikipedia. *Recurrent Neural Network*. URL: `https://en.wikipedia.org/wiki/Recurrent_neural_network`.

[8] Lucas Ou-Yang. *Newspaper3k: Article scraping curation*. URL: `https://newspaper.readthedocs.io/en/latest/`.