Aprendizado de Representações para Processamento de Linguagem Natural

Shih T. Ju[1,2], **Eraldo R. Fernandes**[1],
Irving M. Rodrigues[1], **Augusto Vellozo**[2]

[1]Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

[2]TecSinapse
Tecnologia da Informação LTDA

Resumo. *Relatório de atividades do estágio da primeira autora na empresa TecSinapse no período de agosto de 2014 a agosto de 2016. São apresentados os materiais bibliográficos usados nos estudos preliminares, as tarefas de processamento de linguagem natural abordadas e os algoritmos e modelos usados para solucionar as mesmas. Por fim, a interface dos principais programas desenvolvidos, para treinamento e teste dos modelos, são brevemente descritos.*

1. Introdução

Uma grande quantidade de toda informação disponível atualmente encontra-se na Web sob a forma de textos; seja em livros, artigos, manuais, e-mails, revisões de produtos, *posts* em redes sociais, *blogs*, etc. Apesar desta fonte de recursos ser atrativa e de fácil acesso, a extração automática de informação útil a partir dela é um desafio. Os textos estão escritos em linguagem natural; e o nível de formalidade e estruturação destes textos varia muito, de acordo com o nível de editoração em cada caso.

A área de Processamento de Linguagem Natural (PLN) consiste no desenvolvimento de algoritmos e modelos computacionais para a realização de tarefas que dependem de informações expressas em alguma linguagem natural (por exemplo, tradução e interpretação de textos, busca de informações e interfaces homem-máquina através de conversação). Algumas tarefas clássicas que PLN trata são:

- *POS Tagging.* Etiquetaguem morfossintática de palavras (*part-of-speech tagging*, em inglês) consiste em atribuir a categoria gramatical que cada palavra exerce em uma frase.
- *NER.* Reconhecimento de entidades nomeadas (*named entity recognition*, em inglês) envolve identificar entidades de interesse (pessoas, empresas, locais e data, por exemplo) e atribuir a categoria correta para elas.
- *AS.* Análise de sentimentos busca extrair o sentimento de um texto, por polaridade como positivo, neutro ou negativo.

Numa tarefa como AS, deseja-se que dado um conjunto de textos a máquina consiga classificar o sentimento deste, conforme ilustrado na Figura 1. Esta não é uma tarefa fácil devido a diversos fatores como, por exemplo, sarcasmo, subjetividade, complexidade da linguagem, erros ortográficos. Para termos uma ideia destes problemas, observe os *tweets* a seguir.

- PQ MASTERCHEF É TÃO VICIANTE?

Figura 1. Ilustração da tarefa de análise de sentimentos.

- ESSA MARATONA NA SONY FOI TOP VIU
- Quando a rodada é muito fácil de escalar o santo desconfia
- Issu aí .. https://t.co/n7p3AqJD4E
- E ta moo friacaaa

Este projeto visa elaborar algoritmos baseados em técnicas de *aprendizado de máquina* (AM), mais especificamente aprendizado de máquina supervisionado. Um algoritmo de AM, no caso da tarefa de AS, toma como entrada um conjunto de textos cujos sentimentos são determinados (textos rotulados) e fornece como saída um modelo capaz de resolver a tarefa de AS para textos não rotulados e não vistos anteriormente. O modelo, portanto, identifica padrões na entrada (texto) que indicam, com certa confiabilidade, qual classe (positivo ou negativo) aquele texto pertence. O processo de derivar um modelo através de um algoritmo de AM é denominado *treinamento* e é ilustrado na Figura 2. Portanto, um algoritmo de AM usado para estes fins é denominado *algoritmo de treinamento*.

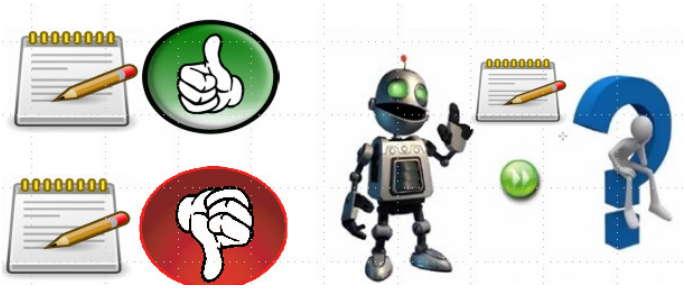

Figura 2. Ilustração do funcionamento do algoritmo de treinamento para a tarefa de análise de sentimentos.

Um exemplo de modelo de aprendizado de máquina é denominado *regressão logística* e é ilustrado na Figura 3. Este modelo é capaz de discriminar uma entrada $\mathbf{x}$ em duas classes (0 ou 1) usando a função linear:

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m,$$

onde $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ representa a entrada através de m atributos numéricos e $\mathbf{w} = (w_0, w_1, w_2, \ldots, w_m)$ são os parâmetros do modelo que ponderam os atributos de entrada. A função discriminante é então dada por:

$$h(\mathbf{x}) = \begin{cases} 1, & \text{se } f(\mathbf{x}) \geq 0 \\ 0, & \text{se } f(\mathbf{x}) < 0. \end{cases}$$

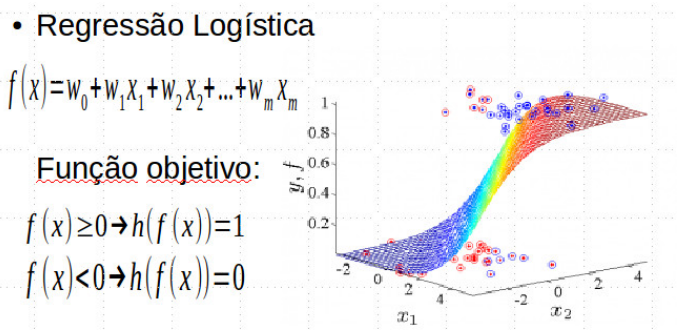

Figura 3. Modelo de regressão logística (classificação binária).

O vetor $\mathbf{w}$ de parâmetros do modelo é determinado pelo algoritmo de treinamento. O objetivo deste algoritmo é separar os exemplos de uma categoria dos exemplos da outra categoria. Na tarefa de AS, por exemplo, deseja-se aprender parâmetros para os quais aplicando-se a função discriminante $h(\mathbf{x})$ consegue-se identificar bem quais textos são positivos e quais são negativos.

No caso de AS, e da maioria das tarefas de PLN, a entrada é composta pelo texto a ser analisado. Entretanto, o algoritmo de treinamento espera uma entrada de tamanho fixo, ou seja, uma entrada composta por m atributos numéricos. Existem diversas maneiras de representar um texto como um vetor de atributos com tamanho fixo. Uma das representações mais comuns é denominada *one-hot encoding*. Nesta representação, o número de atributos m é igual ao número de palavras diferentes consideradas pelo modelo (este número é frequentemente da ordem de algumas centenas de milhares). A cada palavra é então atribuído um número inteiro no conjunto $\{1, 2, \ldots, m\}$; este número identifica a palavra e serve como índice no vetor de atributos. Desta forma, um texto é representado por um vetor $\mathbf{x} = (x_1, x_2, \ldots, x_m)$, onde $x_j = 1$ se a palavra j ocorre no texto e $x_j = 0$ se a palavra não ocorre no texto. É fácil notar que os vetores de entrada $\mathbf{x}$ serão muito esparsos, pois a quantidade de palavras diferentes que ocorrem em um texto é, em geral, bem menor do que a quantidade de palavras de uma linguagem. A representação *one hot* é bastante usada mas apresenta várias limitações. Uma destas limitações é que, no caso do modelo de regressão logística, por exemplo, cada palavra é ponderada por um único parâmetro de maneira independente das outras palavras. Desta forma, o modelo não é capaz de aprender similaridades entre as palavras. Todos os pares de palavras são considerados diferentes da mesma maneira. No caso de AS, isto é um grande problema, pois as palavras *gostei* e *adorei*, por exemplo, deveriam ser consideradas mais similares do que as palavras *gostei* e *odiei*.

Dada a importância da representação da entrada em problemas de AM, existem diversas propostas de representações. O *aprendizado profundo de representações* (*deep learning*) é uma destas propostas e tem atraído muita atenção na última década, devido a resultados impressionantes alcançados em diversas áreas. O aprendizado profundo de representações se baseia em *representações distribuídas* que representam palavras em um espaço que permite a expressão de diversas relações sintáticas e semânticas entre palavras. Além de encontrar uma boa representação, deseja-se que a representação seja multi-nível ou multi-camada (por isto, é chamado profundo). Desta forma, a cada camada, a abstração e a complexidade das representações aumentam. Observe na Figura 4 um exemplo disto

para o problema de reconhecimento de face. A primeira camada representa bordas e contornos. A segunda camada já representa regiões da face, como nariz, olhos e lábios. Finalmente, a última camada é capaz de representar diferentes modelos de rosto. Cada camada de representação toma como entrada a saída da camada anterior.

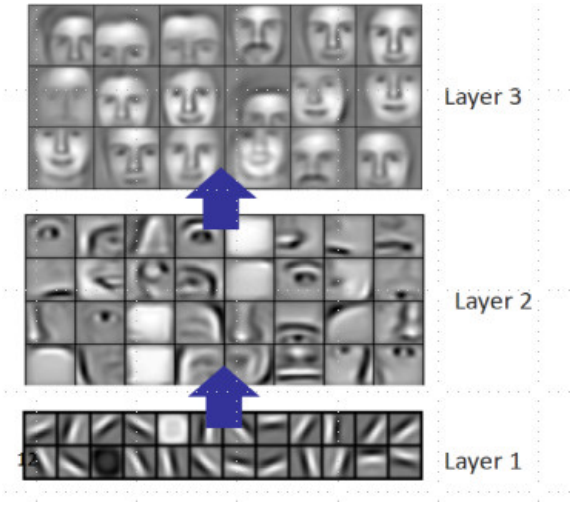

Figura 4. Aprendizado profundo de representações para reconhecimento de face.

Uma característica relevante das representações profundas é que representações intermediárias podem ser compartilhadas por diferentes tarefas, como ilustrado na Figura 5. Em PLN, por exemplo, representações intermediárias poderiam ser compartilhadas entras as tarefas de POS tagging e NER.

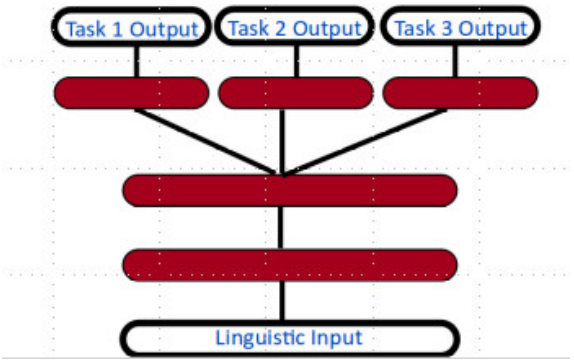

Figura 5. Representações usadas em multi-tarefas

Este projeto foca em algoritmos de aprendizado de máquina para processamento de linguagem natural usando aprendizado profundo de representações. No início do projeto, foram realizados estudos preliminares de diversos conceitos de AM e PLN. Na Seção 2, são sumarizados os estudos preliminares realizados. As tarefas de PLN abordadas neste trabalho são descritas na Seção 3. Na Seção 4, são apresentados conceitos gerais de aprendizado profundo de representações para tarefas de PLN. Na Seção 5, são descritos os modelos usados neste trabalho para as tarefas de PLN. Os principais resultados experimentais obtidos são apresentados na Seção 6. Na Seção 7, são apresentadas

instruções para execução dos principais programas desenvolvidos. A lista a seguir apresenta as atividades que foram realizadas durante o estágio.

1. **Machine Learning - Andrew Ng** [Ng, 2010-2012]
    - Estudo dos tópicos do tutorial
    - Implementação dos exercícios
2. **UFLDL Tutorial** [Ng et al., 2013]
    - Estudo sobre Aprendizado Não Supervisionado de *Features* e *Deep Learning*
    - Implementação dos exercícios
3. **Theano** [Theano Development Team, 2016]
    - Estudo do Tutorial Theano
    - Implementação do Algoritmo de Reconhecimento de Dígitos e *Autoencoder* em Theano
4. **Processamento de Linguagem Natural**
    (a) **Deep Learning for Natural Language Processing (without Magic)** [Socher et al., 2012]
        - Estudo do Artigo [Socher et al., 2013]
        - Implementação do Algoritmo de Reconhecimento de Entidades Nomeadas

    (b) **Natural Language Processing (Almost) from Scratch** [Collobert et al., 2011]
        - Estudo do Artigo
        - Implementação do Algoritmo em Python

    (c) **POS tagging**
        - Leitura do Artigo Learning Character-level Representations for Part-of-Speech Tagging [dos Santos and Zadrozny, 2014b]
        - Estudo do artigo Training State-of-the-Art Portuguese POS Taggers without Handcrafted Features [dos Santos and Zadrozny, 2014a]
        - Implementação do CharWNN

    (d) **Domain Adaptation**
        - DLID: Deep Learning for Domain Adaptation by Interpolating between Domains [Chopra et al., 2013]
        - Fast Easy Unsupervised Domain Adaptation with Marginalized Structured Dropout Features [Yang and Eisenstein, 2014]
        - Adaptação do CharWNN pra DLID

    (e) **Unbiased Concurrent Evaluation on a Budget** [Schnabel et al., 2015]
        - Leitura do artigo
        - Análise de formas para avaliação do modelo

    (f) **Análise de Sentimentos**
        - Leitura do artigo Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts [dos Santos and Gatti, 2014]
        - Implementação de Análise de Sentimentos

2. Estudos Preliminares

Alguns estudos preliminares foram necessários para entender e desenvolver algoritmos de aprendizado de máquina para processamento de linguagem natural. A seguir, são apresentados os conceitos adquiridos e os recursos bibliográficos usados.

2.1. Curso de Aprendizado de Máquina

Existem alguns cursos de aprendizado de máquina disponíveis gratuitamente na internet. O curso ministrado pelo professor Andrew Ng [Ng, 2010-2012] é um dos cursos mais antigos e bem avaliados. As aulas deste curso são voltadas à aplicação de aprendizado de máquina. No decorrer do curso, o aluno tem a oportunidade de implementar algoritmos de AM e ganhar experiência com isso. Familiaridade com programação, álgebra linear básica e probabilidade básica são necessários.

O curso introduz o conceito de aprendizado de máquina (supervisionado e não supervisionado), regressão linear, regressão logística, regularização e algoritmo *naive bayes*. Os exercícios práticos são desenvolvidos na linguagem Octave.

2.2. Tutorial sobre Aprendizado Profundo de Representações

Dentre os tutoriais de aprendizado profundo de representações, destaca-se o tutorial da Universidade de Stanford, preparado pelo prof. Andrew Ng [Ng et al., 2013]. As aulas deste curso são voltadas à aplicação de redes neurais profundas a problemas de reconhecimento de objetos.

Conhecimento dos temas explicitados na seção 2.1 são necessários. Os temas abordados incluem: *autoencoder*, algoritmo de *backpropagation*, técnicas de pré-processamento, softmax, construção de redes neurais profundas para classificação, extração de atributos via convolução de imagens grandes. Dentre os exercícios práticos estão: processamento de imagens para aprendizado de bordas, reconhecimento de número por imagem, entre outros.

2.3. Theano

Dentre diversos frameworks para desenvolvimento de algoritmos de aprendizado de máquina, temos o Theano [Theano Development Team, 2016]. Theano é um framework em Python que permite definir, otimizar e executar expressões matemáticas envolvendo vetores multi-dimensionais de forma simples e eficiente. Possui integração com a biblioteca NumPy e execução de expressões de forma mais rápida devido a geração dinâmica de código em linguagem C.

As vantagens do uso de Theano incluem: velocidade de execução por compilar partes dos grafos de expressão em instruções de CPU ou GPU; cálculo automático de gradiente de expressões simbólicas; e otimizações que usam algoritmos estáveis para calcular expressões numericamente instáveis.

No Theano, usa-se variáveis simbólicas, tal que não possuem valor explícito. Funções são definidas sobre estas variáveis e a computação é feita através de chamadas de função com valores reais. O tutorial da ferramenta descreve os tipos de variáveis que existem, as funções que a biblioteca fornece, como definir novas funções, como calcular gradientes automaticamente, além de exemplificar a implementação de um perceptron multi-camada, que serve de base para a implementação de redes neurais.

3. Processamento de Linguagem Natural

Segundo SBC [2009], a área de Processamento da Linguagem Natural (PLN) lida com problemas relacionados à automação da interpretação e da geração da língua humana em

aplicações como tradução automática, sumarização automática de textos, ferramentas de auxílio à escrita, sistemas de perguntas e respostas, categorização textual, recuperação e extração de informação, entre muitas outras. Além de tarefas relacionadas à criação e disponibilização de dicionários/léxicos e córpus eletrônicos, desenvolvimento de taxonomias e ontologias, investigações em lingüística de córpus, desenvolvimento de esquemas de marcação e anotação de conhecimento lingüístico-computacional, resolução anafórica, análise morfossintática automática, análise semântico-discursiva automática, dentre outras.

Os trabalhos Socher et al. [2012] e Collobert et al. [2011] introduzem os conceitos de aprendizado profundo de representações aplicados à área de processamento de linguagem natural, Nestes trabalhos, são demostrados como textos podem ser representados para computação, e um modelo de rede neural profunda para aplicações de PLN.

3.1. Reconhecimento de Entidades Nomeadas

Reconhecimento de entidades nomeadas (REN) define-se como uma tarefa cujo objetivo é identificar entidades de interesse mencionadas em um texto e classifica-las de acordo com um conjunto de categorias previamente definido. As entidades de interessa são geralmente substantivos próprios, tais como nome de pessoas, organizações e locais; temporais como datas, tempo, dia, ano e mês; entidades numéricas, tais como medições, porcentagens e valores monetários. Exemplos de entidades e suas categorias são apresentados na Tabela 1; e uma frase classificada de acordo com o REN é apresentada na Tabela 2. A base de dados utilizada nesta tarefa foi a apresentada no tutorial de Socher et al. [2013].

Tabela 1. Exemplos de entidades e suas categorias.

Tipo	Etiqueta	Exemplos
Pessoa	PER	Indivíduos, personagens fictícios, pequenos grupos.
Organização	ORG	Empresas, agências, partidos políticos, grupos religiosos.
Localidade	LOC	Bairro, cidade, país.
Outros	O	Nomes de carros, títulos de livros, modelos de computadores.

Tabela 2. Exemplos de NER.

Palavra	João	comprou	um	tênis	na	Netshoes	em	2015	.
Etiqueta	Pessoa	-	-	-	-	Organização	-	Tempo	-

3.2. Part-of-Speech Tagging

Part-of-speech tagging (POS tagging) é o processo de identificar a classe gramatical que cada palavra exerce em uma frase, conforme o exemplo da Tabela 3.

Tabela 3. Exemplos de NER.

Palavra	João	comprou	um	tênis	na	Netshoes	em	2015	.
Etiqueta	Substantivo	Verbo	Artigo	Substantivo	Preposição+Artigo	Substantivo	Preposição	Numeral	Pontuação

Uma das dificuldades desta tarefa é a existência de muitas palavras com diferentes classificações possíveis. Tais palavras podem assumir exercer diferentes classes gramaticais dependendo do contexto em que são empregadas. Por exemplo, na frase *Vamos assistir ao jogo*, a palavra *jogo* é um substantivo que pode significar, dentre outras, uma partida de futebol. Porém, a mesma palavra empregada na frase *Eu jogo videogame* trata-se de uma flexão do verbo *jogar*. A palavra *jogo* é um exemplo de ambiguidade existente na língua portuguesa.

Nesta tarefa, foram utilizadas as bases de dados Mac-Morpho [Aluísio et al., 2003]. Este é um corpus formado por artigos publicados no jornal Folha de São Paulo, em 1994, contendo mais de 1 milhão de palavras, anotadas pelo etiquetador PALAVRAS [Bick, 2000]. Este corpus está atualmente na terceira versão Aluísio et al. [2016].

3.3. Análise de Sentimentos

Análise de sentimentos é uma tarefa cujo objetivo é determinar a polaridade (positiva, negativa ou neutra) da opinião expressa em um texto. Esta tarefa pode ser utilizada para monitorar marcas, produtos ou pessoas em redes sociais e ambientes digitais, por exemplo. Algo essencial para o planejamento estratégico e gestão de crises [Hekima, 2014].

A forma mais comum de classificação de sentimentos é definida pela polaridade. As publicações são divididas entre negativas ou positivas. Em alguns casos, quando a opinião do usuário não é explícita, um texto é considerado da classe neutra. A vantagem da análise polar é a objetividade. Com apenas três opções, a classificação é simplificada e exige menos recursos. No entanto, a análise é limitada a categorias generalizadas e não permite uma compreensão mais profunda de sentimentos dos usuários.

De acordo com Aguiar [2009], algumas dificuldades desta tarefa são:

- textos com erros e frases sintaticamente mal formadas (o que é bastante comum em blogs e redes sociais);
- distinguir se um texto é opinião ou fato e, principalmente, identificar em um fato se existem opiniões embutidas;
- sarcasmos e ironias;
- textos que referenciam mais de um item (iPhone e iPod, por exemplo) com opiniões diferentes sobre os itens;
- uso de pronomes para referenciar itens pode dificultar a identificação de sentenças que mencionam o item de interesse;
- uso de termos informais (gírias, emoticons e abreviações da internet) como, por exemplo, *blz*, *fds* e *:)*;
- propaganda disfarçada, em que blogueiros recebem dinheiro para falar bem de alguma empresa ou produto.

Para esta tarefa foram utilizados tweets do dataset de `http://help.sentiment140.com/for-students/`. Os dados foram pré-processados, tal que os emoticons foram removidos. O arquivo está no formato CSV. Os dados do arquivo possuem 6 campos:

- 0 - a polaridade do tweet (0 = negativo, 2 = neutro, 4 = positivo)
- 1 - o id do tweet (Ex: 2087)
- 2 - a data do tweet (Ex: Sat May 16 23:58:44 UTC 2009)

- 3 - a consulta (Ex: lyx). Se não houve consulta, então o valor é NO_QUERY.
- 4 - o usuário que tweetou (Ex: robotickilldozr)
- 5 - o texto do the tweet (Ex: Lyx is cool)

O dataset possui um arquivo de treino e outro de teste, o arquivo de treino possui 1.6 bilhão de tweets, dos quais 80K foram amostrados para o treino e 16K para dev. Foi utilizado o Streaming API open-source Tweepy para obter tweets em português para aprendizado não supervisionado das representações das palavras com word2vec.

4. Aprendizado Profundo de Representações

A representação dos dados é a definição de como os exemplos de treinamento são representados no algoritmo de aprendizado de máquina. Na maioria dos casos, um exemplo deve ser representado como uma lista, de tamanho fixo, de atributos numéricos. Atributos categóricos ou simbólicos também podem ser utilizados, mas, geralmente, estes atributos são convertidos para atributos numéricos de alguma maneira.

A criação de atributos é frequentemente realizada por especialistas que analisam o problema em questão e sugerem atributos e transformações que melhor representam um exemplo do ponto de vista do algoritmo de AM. Esta é uma maneira lenta, custosa e limitada para se criar representações de dados. Numa tarefa de POS tagging, por exemplo, alguns dos atributos mais utilizados para resolver este problema são: indicação se a palavra começa com letra maiúscula, sufixos da palavra, prefixos da palavra e lema (ou radical) da palavra. Estes atributos foram propostos por especialistas em análise morfossintática e também em aprendizado de máquina. Em análise de sentimentos, por exemplo, podem incluir atributos manuais como dicionários de palavras negativas e positivas, dicionário de negação, entre outros.

Além da criação de atributos manuais ser custosa e lenta, estes atributos tem baixa reutilização. Por exemplo, dicionários de sufixos em português não servem para uma tarefa em inglês. Por isto, o aprendizado automático de representações é uma tarefa tão relevante na área de aprendizado de máquina.

4.1. *One-hot encoding*

Uma das representações mais simples utilizadas para a representação de uma palavra é o *one-hot encoding*. Em que cada palavra é representado por um vetor de 0's menos a posição que identifica o seu id no vocabulário. Ex: suponha V um vocabulário de palavras tal que, $V = \{$ cão, gato, zebra, leão $\}$.

Os vetores que representam essas palavras serão:

- cão = [1, 0, 0, 0]
- gato = [0, 1, 0, 0]
- zebra = [0, 0, 1, 0]
- leão = [0, 0, 0, 1]

Esta é uma representação atômica, bem simples, entretanto o tamanho da representação de cada palavra está diretamente associada ao tamanho do vocabulário utilizado, se tivesse um vocabulário muito grande com 100.000 palavras, as representações teriam 100.000 unidades, o que dificulta a computação. Além disso, o aprendizado dos parâmetros, os quais tentam capturar padrões e comportamentos dos dados de entrada,

não consegue aprender relações entre as palavras, já que cada uma influencia em apenas um dos parâmetros, uma vez que a representação possui como característica, a exclusão mútua.

4.2. Representação distribuída

Na representação distribuída, cada palavra é representado por um vetor de números reais, tal que o tamanho do vetor é arbitrário, e independente do tamanho do vocabulário. Ex: suponha V um vocabulário de palavras tal que, $V = \{$ cão, gato, zebra, leão $\}$, e fixa-se o tamanho do vetor igual a 4.

Os vetores que representam essas palavras podem ser:

- cão = [0.345, 0.234, 0.192, -0.93]
- gato = [0.120, -1.789, 0.102, 0.122]
- zebra = [0.983, 0.211, -0.01, 0.51]
- leão = [0.321, -0.422, 0.289, -0.111]

Os valores destes vetores podem ser inicializados de forma aleatória ou com valores de *word vectors* pré-treinados, que podem ser obtidos com word2vec. Essa representação influencia todos os parâmetros do modelo, permitindo o aprendizado de similaridades entre as palavras.

4.2.1. Word2vec

Word2vec é uma ferramenta que oferece uma implementação eficiente de *bag-of-words* contínua e *skip-gram*, que são usadas na computação dos vetores de representação de palavras. Está disponível em `https://code.google.com/archive/p/word2vec/`, em linguagem C.

Tem como entrada um texto e produz *word vectors* como saída. Ele constrói um vocabulário do texto de treinamento e aprende vetores de representações de palavras, é possível configurar uma série de parâmetros, como, o tamanho dos *word vectors*, o tamanho da janela deslizante, o número mínimo que uma palavra deve aparecer para ser inclusa no vocabulário, entre outros.

Recentemente, foi mostrado que os *word vectors* capturam muitas regularidades linguísticas, como por exemplo, fazendo a operação vetor('Paris') - vetor('France') + vetor('Italy') resulta em um vetor que é muito próximo do vetor('Rome'), e vetor('king') - vetor('man') + vetor('woman') é próximo do vetor('queen').

5. Modelos Implementados

Os algoritmos implementados neste trabalho são baseados na proposta de dos Santos and Gatti [2014]. O aprendizado de representações em nível de palavras (*word vectors*) captura informações sintáticas e semânticas. Enquanto o aprendizado de representações em nível de caracteres captura informações sobre morfologia, forma e capitalização.

A vantagem deste modelo é a diminuição da necessidade de atributos manuais. O aprendizado à nível de palavras tenta capturar relações entre as palavras, já que a entrada é composta por janelas de palavras. Por exemplo, num problema de POS tagging é comum

adjetivos aparecerem depois de substantivos, mas não depois de verbos. O aprendizado à nível de caracteres tenta capturar relações entre os caracteres, que são analisadas por janelas de caracteres. Ex: é comum adjetivos terminarem com sufixo ”ente”, ”ante”, ”ido”, ”osa”, espera-se que o algoritmo possa identificar esses padrões de forma automática.

Este modelo foi aplicado à tarefa de POS tagging e análise de sentimentos. Para POS tagging, o dataset de treino e teste são os textos do corpus Mac-Morpho citados na subseção 3.2, para análise de sentimentos, o dataset de treino e teste são os tweets conforme descrito na seção 3.3. Os dados de entrada são pré-processados conforme a seção 5.1, e pode-se utilizar o modelo WNN ou CharWNN.

5.1. Pré-processamento

O pré-processamento para ambas as tarefas envolveu a tokenização descrita na subseção 5.1.2, o processo de substituição, para POS tagging envolveu apenas a substituição de dígitos para 0, enquanto que para análise de sentimentos aplicaram-se todas as substituições descritas na subseção 5.1.1, devido à especificidade de algumas das substituições voltadas aos textos de *twitter*.

5.1.1. Substituição de palavras

Os dígitos são todos substituídos por 0, os links dentro dos textos são substituídos por uma palavra especial «url» e os nomes de usuários, por «user». Conforme a figura 6.

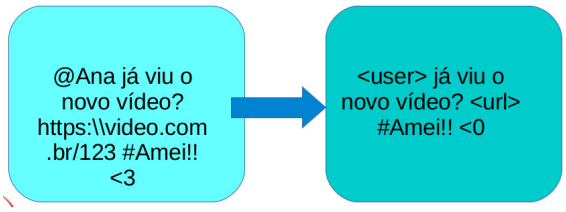

Figura 6. Exemplo de substituição de palavras

5.1.2. Tokenização

Tokenização é o processo de dividir o texto de entrada em unidades chamadas tokens. Cada token representa uma palavra ou algo como um número ou um sinal de pontuação. O resultado desse processo é uma sequência de palavras intercaladas por espaços ou por símbolos delimitadores. Conforme figura 7.

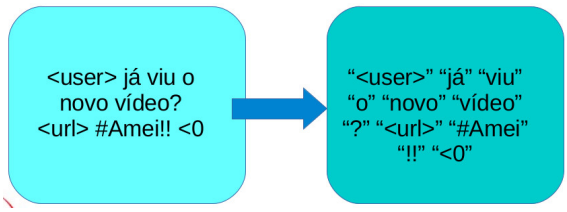

Figura 7. Exemplo de substituição de palavras

O tokenizador utilizado foi desenvolvido pela Universidade de Stanford, disponível em `http://nlp.stanford.edu/software/tokenizer.shtml`, que insere espaços entre tokens, ou caracteres de fim de linha, dependendo da configuração.

5.2. Arquitetura da Rede Neural

Dada uma frase, o modelo calcula a probabilidade dela ser de cada uma das categorias existentes da entrada. E passa por uma série de camadas, as quais atributos com nível crescente de abstração e complexidade são extraídas. A rede extrai de atributos à nível de caracteres a atributos à nível de frase.

5.2.1. WNN

Para cada frase de entrada são montadas janelas de palavras, se o tamanho da janela for *k* e a frase conter *n* palavras, então serão montadas *n* janelas, cada uma centrada em cada uma das palavras da frase, tal que cada janela é formada pela palavra e seus (*k*-1)/2 vizinhos à esquerda e à direita. Ex: seja *k* igual a 3, conforme a figura 8, quando não possuir (*k*-1)/2 vizinhos, o que estiver faltando é preenchido com uma palavra especial.

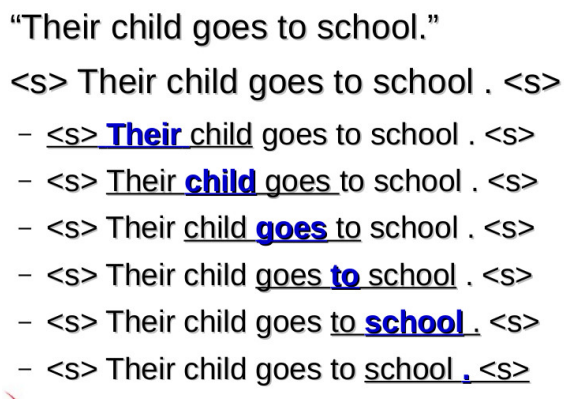

Figura 8. Exemplo de janelas em uma frase, em sublinhado as janelas, e em azul a palavra alvo (central).

Cada palavra dentro da janela é mapeada à um *word vector* que o representa, através da consulta de uma tabela que contém a cada palavra do vocabulário um *word vector* associado, a *Word Lookup Table*, conforme explicitado na figura 9.

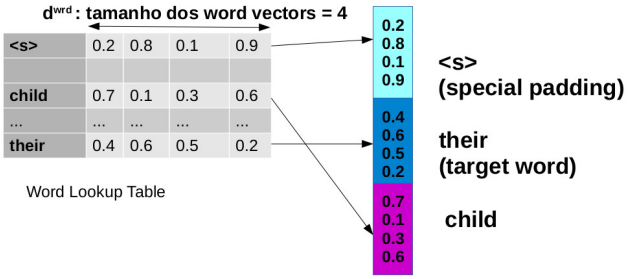

Figura 9. Ilustração de uma janela de palavras e sua relação com a *Word Lookup Table*.

O modelo WNN para POS tagging é ilustrado na figura 10. Para cada palavra da frase, é montada sua janela correspondente, cada palavra da janela é mapeada ao seu *word vector* correspondente através da consulta da *Word Lookup Table*, a saída desta camada é a entrada para a camada oculta e depois passa por uma camada de *softmax* que calcula a probabilidade pra cada classe gramatical.

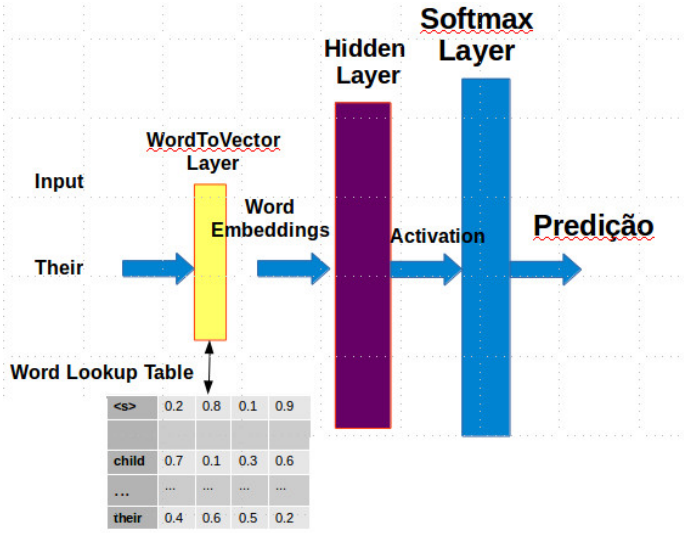

Figura 10. Ilustração do modelo WNN para POS tagging

O modelo WNN para análise de sentimentos é ilustrado na figura 11. Dado como entrada uma frase, são montadas as janelas de palavras, cada palavra é mapeada ao seu *word vector* correspondente através da consulta da *Word Lookup Table*, a saída desta camada é a entrada para a camada convolucional de frase que busca extrair os atributos à nível de frase, a saída desta alimenta uma camada oculta e depois passa por uma camada de *softmax* que calcula a probabilidade pra cada classe (positiva, neutra, negativa).

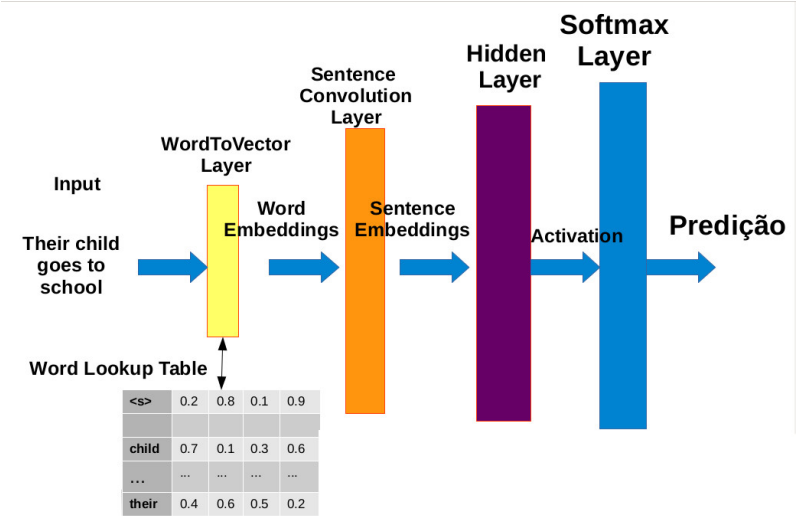

Figura 11. Ilustração do modelo WNN para análise de sentimentos

A camada de convolução de palavras é ilustrada na figura 12, após montar as janelas de palavras da frase, passa-se por uma matriz de parâmetros W^1, que serve como um filtro de atributos, sobre a saída desta passa-se uma função de *max*, para capturar os atributos de maior peso, para representar a frase, e esta representação alimenta a camada oculta.

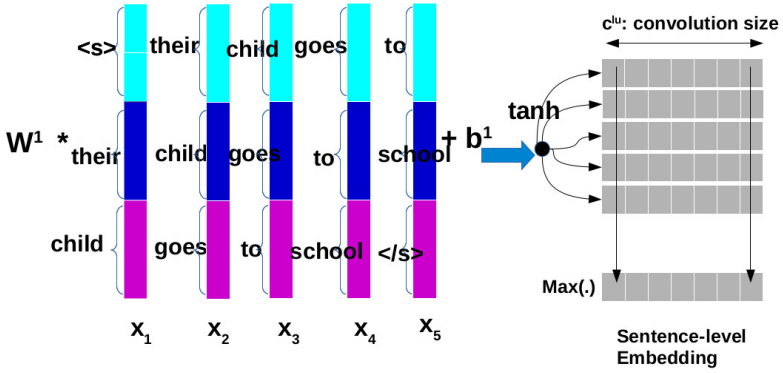

Figura 12. Ilustração da camada de convolução de frase do modelo WNN para análise de sentimentos

5.2.2. CharWNN

Para cada palavra da entrada são montadas janelas de caracteres, se o tamanho da janela for *k* e a palavra conter *n* caracteres, então serão montadas *n* janelas, cada uma centrada em cada um dos caracteres da palavra, tal que cada janela é formada pelo caracter e seus (*k*-1)/2 vizinhos à esquerda e à direita, de forma similar à janela de palavras. Quando não possuir (*k*-1)/2 vizinhos, o que estiver faltando é preenchido com caracter especial.

Cada caracter dentro da janela é mapeada à um *char vetor* que o representa, através da consulta de uma tabela que contém a cada caracter do vocabulário de caracteres um *char vector* associado, *Char Lookup Table*, conforme explicitado na figura 13.

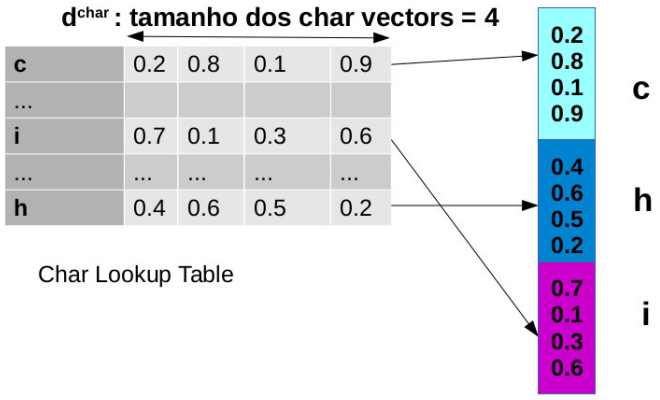

Figura 13. Ilustração de uma janela de caracteres e sua relação com *Char Lookup Table*.

A camada de convolução de caracteres é similar a de palavras. É ilustrada na figura 14, após montar as janelas de caracteres das palavras, passa-se por uma matriz de parâmetros W^0, que serve como um filtro de atributos, sobre a saída desta passa-se uma função de *max*, para capturar os atributos de maior peso, para representar a palavra.

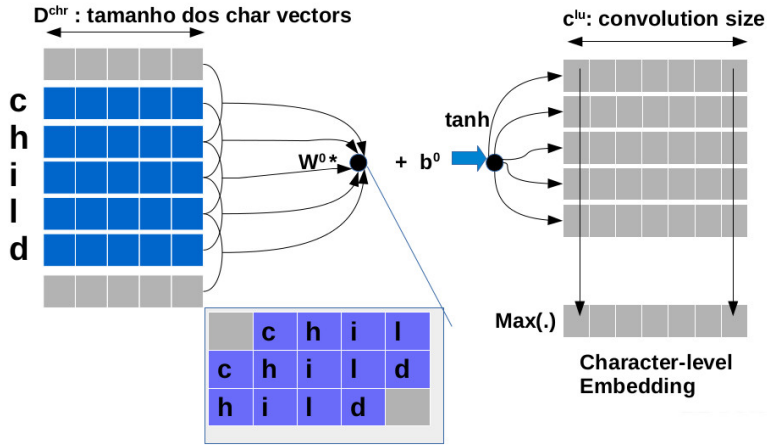

Figura 14. Ilustração da camada de convolução de caracteres

O modelo CharWNN para análise de sentimentos é ilustrado na figura 15. Dado como entrada uma frase, são montadas as janelas de palavras, cada palavra é mapeada ao seu *word vector* correspondente através da consulta da *Word Lookup Table*, além disso são montadas as janelas de caracteres, cada caracter da palavra é mapeada ao seu *char vector* correspondente através da consulta de *Char Lookup Table*, aplica-se a convolução sobre essas janelas de caracteres, essas duas representações são concatenadas e alimentam a camada convolucional de frase que busca extrair os atributos à nível de frase, a saída desta alimenta uma camada oculta e depois passa por uma camada de *softmax* que calcula a probabilidade pra cada classe.

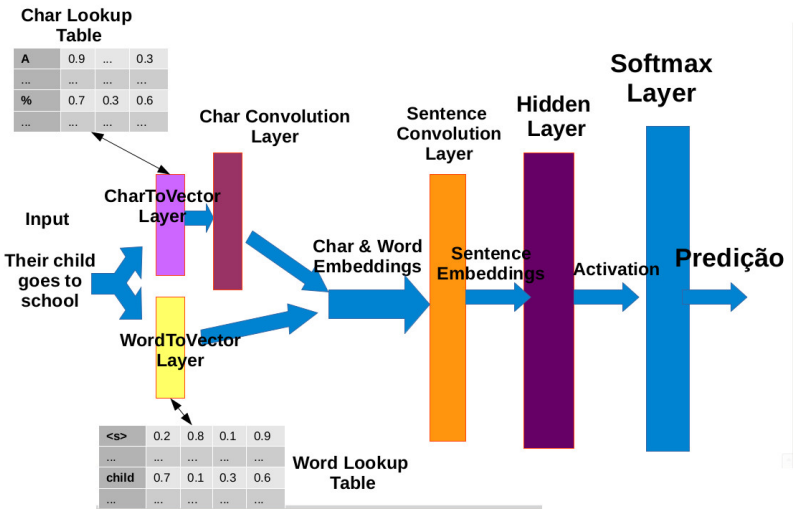

Figura 15. Ilustração do modelo CharWNN para análise de sentimentos

O modelo de CharWNN para POS tagging é similar ao da figura 15, a diferença está na ausência da camada de convolução da frase.

6. Resultados Experimentais

6.1. POS tagging modelo por frase sobre Mac-Morpho V1

A tabela 4 contém o valor das acurácias obtidas do modelo por frase[1] sobre o *dataset* do Mac-Morpho. O resultado OOSV (Out-of-Supervised-Vocabulary) representa a acurácia calculada sobre as palavras que não estão no treino, mas que aparecem no *dataset* de teste. E, o resultado OOUV (Out-of-Unsupervised-Vocabulary) representa a acurácia calculada sobre as palavras que estão fora do vocabulário de treino ou do vocabulário pré-treinado usado, mas que aparecem no *dataset* de teste.

Tabela 4. Resultado do POS tagging modelo por frase

Sistema	All Words	OOSV	OOUV
Cícero's CharWNN	97.47	92.49	89.74
Ours CharWNN	96.94	89.68	85.56
Cícero's WNN	96.19	83.08	75.40
Ours WNN	95.83	82.00	71.71

6.2. Análise de Sentimentos para Redes Sociais (Twitter) em Português

A tabela 5 contém o valor das acurácias obtidas aplicando o modelo sobre um *dataset* do ctrls, que contém 322 *tweets*. Foi realizado validação cruzada de 6 folds para obter estes resultados, a classificação dos tweets foi: positivo, neutro ou negativo. Tal que no nosso modelo foram utilizados *word vectors* pré-treinados com word2vec, e aplicados uma normalização.

Tabela 5. Resultado para dataset em português

Modelo	acurácia	desvio padrão
WNN (nosso)	73,17%	$\pm 3,49\%$
CharWNN (nosso)	74,397%	$\pm 3,472\%$
IMBHN (Rafael)	61,7%	$\pm 4,368\%$

6.3. Análise de Sentimentos para Redes Sociais (Twitter) em Inglês

A tabela 6 contém o valor das acurácias obtidas aplicando-se o modelo sobre um *dataset* de *tweets* em inglês citado na seção 3.3. A classificação dos *tweets* foi: positivo ou negativo.

O modelo do Cícero é o estado da arte atual, ele utilizou *word vectors* pré-treinados usando textos da Wikipedia, não foi possível obter estes dados, logo usamos os *word vectors* disponíveis em `http://nlp.stanford.edu/projects/glove/`.

[1] O modelo por frase encontra-se com *bug*.

Tabela 6. Resultado para dataset em inglês

Modelo	Aleatório	Wiki	Glove_wiki
Cícero's CharWNN*	81.9	86.4	-
Ours CharWNN	81.62	-	83.84
Cícero's WNN	82.2	85.2	-
Ours WNN	82.45	-	82.73
LProp Speriosu et al. [2011]	84.7		
MaxEnt Go et al. [2009]	83.0		
NB Go et al. [2009]	82.7		
SVM Go et al. [2009]	82.2		

7. Programas Principais

O código está disponível no repositório `pln-ufms` no Bitbucket[2]. A versão do código usado neste trabalho está na tag `v0.2`.

Para rodar o modelo de POS tagging basta usar o comando:

OMP_NUM_THREADS=1 nohup python -u Postag.py [–args] » arquivo_saida.txt 2»&1 &

OMP_NUM_THREADS define o número de threads que deseja utilizar. O arquivo_saida.txt é o arquivo em que será direcionado a saída com as acurácias de teste impressas. Enquanto para análise de sentimentos:

OMP_NUM_THREADS=1 nohup python -u SentimentAnalysis.py [–args] » arquivo_saida.txt 2»&1 &

Sobre os argumentos da rede basta consultar os arquivos Postag.py ou SentimentAnalysis para mais detalhes.

Exemplo de execução de WNN para AS:

OMP_NUM_THREADS=1 nohup python -u SentimentAnalysis.py –train=/caminho_do_arquivo_de_treino/arquivo_treino.txt –test=/caminho_do_arquivo_de_teste/arquivo_de_teste.txt –numepochs 15 –senlayerwithact –hiddenlayersize=300 –charConvolutionalLayerSize=50 –wordConvolutionalLayerSize=300 –wordWindowSize=5 –charWindowSize=3 –numperepoch 1 –lr=0.01 –wordVecSize=50 –charVecSize=5 –maxSizeOfWord=30 –startSymbol «s» –endSymbol «s» –filters DataOperation.TransformNumberToZeroFilter TransformNumberToZeroFilter DataOperation.RemoveURL RemoveURL DataOperation.RemoveUserName RemoveUserName –testoosv –testoouv –lrupdstrategy=divide_epoch –unknownwordstrategy=”mean_vector--mean_size 1000 –seed=1443901331 –charVecsInit=’randomAll’ –vocab=/caminho_do_vocabulario_com_word_vectors/vocabulario.txt » arquivo_saida.out 2»&1 &

[2]https://bitbucket.org/tecsinapse/pln-ufms

O argumento –numepochs define o número de épocas que será rodado sobre o arquivo de treino. O argumento –senlayerwithact ativa a função de ativação para a camada convolucional da frase. Os argumentos –hiddenlayersize, –charConvolutionalLayerSize, –wordConvolutionalLayerSize, –wordWindowSize, –charWindowSize, –wordVecSize, –charVecSize, são parâmetros da rede que quando não especificados utilizam o valor *default* definido no programa.

Para ativar o CharWNN basta incluir: –withCharwnn. É recomendado usar função de ativação na camada de convolução de caracteres através de: –charwnnwithact.

Para usar validação cruzada basta incluir: –crossvalidation e –kfold=6 , neste exemplo configurou-se *fold* de tamanho 6.

Para salvar o modelo treinado basta incluir: –saveModel=/caminho_do_arquivo_onde_sera_salvo_o_modelo/nome_modelo.

Caso queira rodar um arquivo de teste sobre um modelo salvo, basta incluir: –loadModel=/caminho_do_modelo/nome_modelo.

7.1. Programa de Análise de Sentimentos

A melhor acurácia obtida por validação cruzada de 6 *folds*, foi obtida com a calibração dos parâmetros sobre o dataset de *tweets* em português, com normalização inicial sobre os *word vectors* pré-treinados usando o word2vec. Para o modelo CharWNN o coeficiente de normalização utilizado foi 1.0 e para WNN 0.5.

Na pasta Examples do diretório do projeto encontram-se:

- models: subpasta que contém os modelos de CharWNN e de WNN.
  - ctrls: modelos para *tweets* em português;
  - twitter_en: modelos para *tweets* em inglês.
- dataset: conjunto de dados de treino e teste, e *word vectors* pré-treinados.
  - ctrls:
    - ctrls.tokenized.txt_x_6_folds: os arquivos que foram utilizados na validação cruzada de 6 *folds*.
    - ctrls.tokenized.txt: dataset do ctrls com todos os 322 *tweets* (positivo, negativo e neutro).
    - twi_pt_win5_min_20_siz_30_sg_1_neg_5.txt: *word vectors* pré-treinados no word2vec e utilizados na validação cruzada.
  - twitter_en:
    - train.80000.tokenized.csv: dataset de treinamento com 80.000 *tweets* em inglês.
    - dev.16000.tokenized.csv: dataset de dev, utilizado para teste durante o treino, com 16.000 tweets em inglês.
    - testdata.only.pos.neg.csv: dataset de teste.
    - twi_full_wordvec_win9_iter_25_min_10_size_50_sg_1_neg_5.txt: exemplo de *word vectors* pré-treinados para *tweets* em inglês.
- output: subpasta com exemplos de saída do algoritmo.
- README.txt: orientações de como executar o algoritmo com e sem script.
- train_model.sh: script para treinar modelos.
- test_model.sh: script para testar modelos.

Exemplo de treino de modelo usando script train_model.sh:

./train_model.sh -train=dataset/ctrls/ctrls.tokenized.txt_x_6_folds/train_1.txt -test=dataset/ctrls/ctrls.tokenized.txt_x_6_folds/test_1.txt -prediction=predicao.txt -model=models/treino_wnn.model -out=saida.out –as –wnn -iter=2 -vocab=dataset/ctrls/twi_pt_win5_min_20_siz_30_sg_1_neg_5.txt -lr=0.01 –wnn

Tal que:

- -train: arquivo que vai ser utilizado para o treino do modelo (obrigatório)
- -test: arquivo de teste (obrigatório)
- -prediction: arquivo onde deseja salvar a predição do teste
- -model: arquivo que deseja salvar o modelo
- -out: arquivo onde deseja imprimir a saída (com os valores de acurácia)
- -iter: número de iterações
- -vocab: vocabulário para o treino
- -cross: número de folds para validação cruzada (a validação é ativada com este parâmetro)
- -lr: taxa de aprendizado (default 0.01)

Os parâmetros de ativação:

- –as : o algoritmo que será executado é a análise de sentimentos
- –pos : o algoritmo que será executado é o POS tagging
- –wnn: o modelo wnn será usado
- –charwnn: o modelo charwnn será usado
- –oouv : calcula acurácia sobre as palavras fora do vocabulário de treino
- –oosv : calcula acurácia sobre as palavras fora do treino

Para testar um modelo usando o script test_model.sh:

./test_model.sh -test=dataset/ctrls/ctrls.tokenized.txt_x_6_folds/test_1.txt -prediction=predicao.txt -model=models/ctrls/ctrls_wnn.model -out=saida.out –as

Tal que:

- -train: arquivo que foi utilizado para o treino do modelo (requerido quando usa –oouv ou –oosv)
- -test: arquivo de teste (obrigatório)
- -prediction: arquivo onde deseja salvar a predição do teste
- -model: o modelo que deseja testar (obrigatório)
- -out: arquivo onde deseja imprimir a saída (com o valor da acurácia)

Na subpasta Progs, encontram-se o .jar da stanford usado para tokenizar, e o executável word2vec pra treinar os *word vectors*, além de um README.txt exemplificando o uso.

Exemplo de tokenização:

nohup java -mx200m -cp stanford-postagger.jar edu.stanford.nlp.process.PTBTokenizer -preserveLines exemplo.txt ≫ exemplo_tokenizado.txt 2≫&1 &

Tem como entrada o arquivo exemplo.txt, o argumento -preserveLines faz com que no arquivo de saída haja a preservação das linhas. O arquivo de saída exemplo_tokenizado.txt gerado possui na primeira linha e na última linha, informações referentes ao tempo e número de tokens que podem ser deletados.

Exemplo de treinamento de *word vectors*:

nohup ./word2vec -train exemplo_tokenizado.txt -output exemplo_wordvec.txt -window 3 -threads 20 -iter 25 -min-count 2 -size 30 -sg 1 -negative 5 -seed 1443901331 2≫&1 &

O arquivo de entrada exemplo_tokenizado.txt (-train) contém as frases sobre as quais deseja treinar os *word vectors* e o arquivo exemplo_wordvec.txt (-output) onde esses *word vectors* serão salvos. Configurou-se com janela de palavras de tamanho 3 (-window), com 20 threads executando (-threads), rodando 25 iterações (-iter), tal que cada palavra deve aparecer no mínimo 2 vezes para ser inclusa no vocabulário (-min-count), o tamanho dos *word vectors* foi fixado em 30, usando o algoritmo skip-gram (-sg), e *negative sampling* (-negative).

Existem outras configurações que podem ser exploradas.

Referências

Fabrício Aguiar. Mineração de opinião, 2009. URL http://blog.vettalabs.com/2009/10/26/mineracao-de-opiniao/.

Sandra Aluísio, Jorge Pelizzoni, Ana Raquel Marchi, Lucélia de Oliveira, Regiana Manenti, and Vanessa Marquiafável. An account of the challenge of tagging a reference corpus for brazilian portuguese, 2003. URL http://link.springer.com/chapter/10.1007/3-540-45011-4_17.

Sandra Aluísio, Jorge Pelizzoni, Ana Raquel Marchi, Lucélia de Oliveira, Regiana Manenti, and Vanessa Marquiafável. Mac-morpho, 2016. URL http://nilc.icmc.usp.br/macmorpho/.

Eckhard Bick. The parsing system palavras, automatic grammatical analysis of portuguese in a constraint grammar framework. *Aarhus University Press*, 2000.

Sumit Chopra, Suhrid Balakrishnan, and Raghuraman Gopalan. Dlid: Deep learning for domain adaptation by interpolating between domains, 2013. URL http://deeplearning.net/wp-content/uploads/2013/03/dlid.pdf.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch, 2011. URL http://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf.

Cícero Nogueira dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts., 2014. URL https://www.semanticscholar.org/paper/Deep-Convolutional-Neural-Networks-for-Sentiment-Santos-Gatti/b0aca3e7877c3c20958b0fae5cbf2dd602104859/pdf.

Cícero Nogueira dos Santos and Bianca Zadrozny. Training state-of-the-art portuguese pos taggers without handcrafted features, 2014a. URL http://link.springer.com/chapter/10.1007/978-3-319-09761-9_8.

Cícero Nogueira dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging., 2014b. URL http://www.jmlr.org/proceedings/papers/v32/santos14.pdf.

Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision, 2009. URL http://s3.

amazonaws.com/academia.edu.documents/34632156/Twitter_Sentiment_Classification_using_Distant_Supervision.pdf?AWSAccessKeyId=AKIAJ56TQJRTWSMTNPEA&Expires=1470289421&Signature=r8U3r%2FONXL6%2BSBGEP5Pfxa6mRIU%3D&response-content-disposition=inline%3B%20filename%3DTwitter_Sentiment_Classification_using_D.pdf.

Hekima. Como classificar sentimentos nas redes sociais?, 2014. URL http://www.bigdatabusiness.com.br/como-classificar-sentimentos-nas-redes-sociais/.

Andrew Ng. Machine learning, 2010-2012. URL http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning.

Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, and Caroline Suen. Ufldl tutorial, 2013. URL http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial.

SBC. Processamento de linguagem natural, 2009. URL http://www.sbc.org.br/14-comissoes/394-processamento-de-linguagem-natural.

Tobias Schnabel, Adith Swaminathan, and Thorsten Joachims. Unbiased concurrent evaluation on a budget, 2015. URL http://doi.acm.org/10.1145/2740908.2742565.

Richard Socher, Yoshua Bengio, and Christopher D Manning. Deep learning for nlp (without magic), 2012. URL http://nlp.stanford.edu/courses/NAACL2013/.

Richard Socher, Yoshua Bengio, and Christopher D Manning. Deep learning for nlp (without magic), 2013. URL http://nlp.stanford.edu/courses/NAACL2013/NAACL2013-Socher-Manning-DeepLearning.pdf.

Michael Speriosu, Nikita Sudan, Sid Upadhyay, and Jason Baldridge. Twitter polarity classification with label propagation over lexical links and the follower graph, 2011. URL http://dl.acm.org/citation.cfm?id=2140465.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions, May 2016. URL http://arxiv.org/abs/1605.02688.

Yi Yang and Jacob Eisenstein. Fast easy unsupervised domain adaptation with marginalized structured dropout., 2014. URL aclweb.org/anthology/P/P14/P14-2088.pdf.