



Universidade Federal de Pernambuco

Centro de Informática

Graduação de Ciência da Computação

Detecção de Posicionamento no contexto de *Fake News*

Larissa Navarro Passos de Araujo

Trabalho de Graduação

Recife

Junho de 2017

Universidade Federal de Pernambuco
Centro de Informática

Larissa Navarro Passos de Araujo

Detecção de Posicionamento no contexto de *Fake News*

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Tsang Ing Ren

Recife

Junho de 2017

Agradecimentos

Primeiramente, gostaria de agradecer à Universidade Federal de Pernambuco e ao Centro de Informática, e a todos os seus funcionários, por todo o apoio e infraestrutura que me ajudaram a desenvolver e completar meus estudos com tamanha qualidade. Minha experiência de intercâmbio no Ciência sem Fronteiras me fez ver o quão boa é a qualidade do trabalho desenvolvido no centro.

Gostaria de agradecer a todos os professores que me ensinaram e orientaram durante o curso. Em especial, ao professor Dr. Tsang Ing Ren, por ter me orientado e acompanhado desde o primeiro período, ajudando a fomentar em mim o interesse pela Inteligência Artificial. Também gostaria de agradecer ao professor Dr. Luciano Barbosa, a João Pedro de Carvalho Magalhães e Giovanni Carvalho, que junto ao professor Tsang, colaboraram e participaram de maneira decisiva do *Fake News Challenge*. O bom resultado que obtivemos em tão pouco tempo de preparação mostra a excelência deste trabalho.

Minha gratidão também a todos que fazem parte da In Loco Media, empresa na qual trabalho desde janeiro de 2016, e me fez ver que é possível sim criar e manter tecnologias disruptivas que afetam milhões de pessoas em solo pernambucano. Especialmente, agradeço a Gabriel Falcone, que sempre me incentivou e acreditou no meu potencial, e à equipe de Mobile, cuja convivência diária é sempre um grande aprendizado.

Agradeço também ao PET Informática, e ao tutor professor Dr. Fernando da Fonseca de Souza. Os quase dois anos que passei como parte do grupo me ajudaram a evoluir não só academicamente, mas também como pessoa. Além disso, gostaria de agradecer muito à minha turma. Fazer parte de um grupo tão inteligente, dedicado e colaborativo foi essencial para todo o sucesso que colhemos e colheremos no futuro. Além disso, aos amigos que fiz para o resto da vida, em especial Leonardo Andrade, Maria Gabriela Cardoso, Marina Haack, Vinícius Cousseau. Agradeço também aos meus amigos de colégio que me acompanham até hoje, Wellington Oliveira e Bianca Ximenes.

Finalmente, agradeço a toda minha família por todo o apoio proporcionados durante a vida. Especialmente à minha mãe, Valéria, minha irmã Pollyana e aos meus dois pais, André e Gladiston, que infelizmente não estão mais aqui. Se eu consegui chegar até aqui e conquistar tudo que consegui foi por conta deles.

Resumo

A última campanha presidencial americana trouxe à tona o quão grave o problema de *fake news* havia se tornado. A desconfiança da população em relação às grandes organizações de notícias aliadas a redes sociais transformadas em câmaras de eco formou um ambiente propício para a disseminação de notícias falsas, sem a devida verificação. A detecção de *fake news* é uma tarefa extremamente complicada, mesmo para humanos, e a automatização de tal problema ainda está distante. Um primeiro passo nesse sentido é a automatização da detecção de posicionamento de textos em relação a outros, a base do *Fake News Challenge*, a primeira competição de Processamento de Linguagem Natural na área. O objetivo deste trabalho é implementar um sistema de classificação de posicionamento de um corpo de texto em relação a uma manchete. O texto pode concordar, discordar, discutir ou não ser relacionado à manchete. Para isso, serão utilizadas técnicas clássicas de Aprendizagem de Máquina e Processamento de Linguagem Natural.

Palavras-chave: processamento de linguagem natural, aprendizagem de máquina, engenharia de características.

Abstract

The last American presidential run has brought the Fake News problem to the light. The growing weariness against the great media conglomerates, along with the echo chambers of social networks, has developed an ideal environment for the spreading of Fake News. Fake News detection is an extremely difficult task, even for humans. As such, automation is still a long way away. Stance Detection automation is a first step in that Direction. It is the basis of the Fake News Challenge, the first Natural Language Processing competition in this field. The goal of this work is to implement a stance detection system, capable of classifying the stance of a body of text relative to a headline. The text can agree, disagree, discuss or be unrelated to the headline. Classical Machine Learning and Natural Language Processing techniques will be utilized.

Keywords: machine learning, natural language processing, feature engineering.

Sumário

1.	Introdução	1
1.1	Objetivos	2
1.2	Estrutura do Trabalho	2
2.	Detecção de Posicionamento	3
2.1	Trabalhos Recentes	4
3.	Técnicas Utilizadas	6
3.1	Classificadores	6
3.1.1	Máquinas de Vetores Suporte (SVMs)	6
3.1.2	Classificador Gradient Tree Boosting	7
3.1.3	Classificador Random Forest	8
3.2	Características	9
3.2.1	Interseção de palavras	9
3.2.2	Palavras de refutação	9
3.2.3	Polaridade	10
3.2.4	Co-ocorrência	10
3.2.5	Bag of Words	10
3.2.6	Term Frequency – Inverse Document Frequency (TF-IDF)	11
3.2.7	Cosseno	11
3.2.8	Part-of-speech Tagging	12
3.2.9	Extração de tópicos	12
3.2.10	Proporção de antônimos	12
3.2.11	Word2vec	13
4.	Fake News Challenge	14
4.1	Arquitetura	16
4.2	Submissão	17
5.	Resultados	18
5.1	Conjunto de Treinamento	18
5.2	Conjunto de Testes	19
6.	Conclusão	22
6.1	Trabalhos Futuros	23
	Apêndice A – Implementação das features do sistema	24
	Apêndice B – Script de treinamento	30
	Apêndice C – Script de submissão	31
	Referências Bibliográficas	33

1. Introdução

Desde a última campanha presidencial americana o termo *fake news* tornou-se um grande problema tanto para a indústria de notícias, como para grandes redes sociais como *Facebook*¹ e *Twitter*². O termo é aplicável a uma matéria inventada com a intenção de ludibriar o público (NEW YORK TIMES, 2016). Uma pesquisa feita em dezembro de 2016 revelou que 64% dos adultos norte-americanos disseram ter se sentido muito confusos sobre eventos correntes, devido a notícias inventadas (PEW RESEARCH, 2016).

Tanta incerteza quanto à veracidade de informações levou ao crescimento de organizações de *fact checking*, ou seja, a tarefa de determinar a veracidade e corretude de afirmações textuais. Ela é extremamente complexa, e mesmo especialistas enfrentam dificuldades ao executá-la. Assim, seria de grande valia para diversos setores uma versão automatizada de *fact checking*. No entanto, este problema ainda está muito longe de ser resolvida no campo do Processamento de Linguagem Natural. Desafios existentes incluem a necessidade de aplicação de contexto, adquirir e verificar fontes de verdade, falta de dados de treinamento, entre outros (FACTMATA, 2016).

Neste contexto, surgiu o *Fake News Challenge*³, uma competição internacional na área de Inteligência Artificial, que busca resolver a primeira parte do problema de *fact checking*: definir o posicionamento de dois textos em relação a um assunto, afirmação ou problema, a chamada detecção de posicionamento. Na competição, o problema foi reduzido à tarefa de detectar o posicionamento de um corpo de texto de notícia relativo à uma manchete. O corpo de texto poderia concordar, discordar, discutir ou não ser relacionado à manchete.

Uma boa solução para o problema de detecção de posicionamento permitiria a um *Fact Checker* humano submeter uma manchete e recuperar todos os textos com posicionamentos relativos a esta manchete, facilitando seu trabalho de recuperar fontes relevantes. Pode-se usar detecção de posicionamento também para classificar uma

¹ <https://www.facebook.com>

² <https://www.twitter.com>

³ <http://www.fakenewschallenge.org>

matéria como falsa ou verdadeira, utilizando a credibilidade ponderada aplicada ao posicionamento de outras fontes sobre o assunto.

1.1 Objetivos

O objetivo deste trabalho é construir um sistema que possa classificar o posicionamento relativo entre um corpo de texto e uma manchete, utilizando técnicas clássicas de Aprendizagem de Máquina e Processamento de Linguagem Natural, a fim de participar do *Fake News Challenge*. Para tanto, pretende-se utilizar os conjuntos de dados de treinamento e teste fornecidos pela competição, de maneira a comparar o desempenho de diferentes classificadores e características.

1.2 Estrutura do Trabalho

Este trabalho está dividido em 6 capítulos, incluindo este capítulo introdutório. O capítulo 2 introduzirá o problema da detecção de posicionamento, apresentando suas características principais e o atual estado da arte. Já o capítulo 3 terá como enfoque as técnicas de Aprendizagem de Máquina e Processamento de Linguagem Natural utilizadas durante o desenvolvimento do projeto. O capítulo 4 discorrerá sobre o problema específico da competição, e a arquitetura projetada e implementada para resolvê-lo.

Por fim, no capítulo 5 são apresentados os resultados obtidos e, no capítulo 6, as conclusões alcançadas e possíveis melhorias e desafios futuros.

2. Detecção de Posicionamento

A proliferação de redes sociais possibilitou aos seus usuários meios pelos quais explicitarem suas opiniões e posicionamentos sobre os mais diversos temas. A impossibilidade de analisar tal volume de texto manualmente trouxe uma grande visibilidade à área de Processamento de Linguagem Natural, especificamente à análise de sentimentos.

A análise de sentimentos tem como objetivo analisar um texto e extrair sentimentos a partir dele, geralmente em relação a alguma entidade, como por exemplo, identificar satisfação do cliente em relação a um produto a partir de uma crítica ao mesmo. Dentro da análise de sentimentos existem várias tarefas distintas, como por exemplo a detecção de polaridade, extração de sentimentos, identificação de subjetividade ou objetividade de textos e outros.

Detecção de posicionamento é uma tarefa relacionada à análise de sentimentos. Ela tem como objetivo determinar, a partir de um texto, se seu autor é favorável, contra ou neutro em relação a uma proposição ou alvo. Este alvo pode ser uma pessoa, organização, política governamental, um produto, etc. Além disso, a proposição ou o alvo pode não estar explicitamente definido no texto (MEISELMAN, 2016). O Quadro 1 mostra exemplos de detecção de posicionamento em diferentes contextos.

Quadro 1: Exemplos de detecção de Posicionamento

Texto	Alvo	Posicionamento
“As grávidas são mais que incubadores ambulantes. Elas também possuem direitos”	Legalização do aborto	Favorável
“Hillary Clinton possui alguns pontos fortes e algumas fraquezas”	Hillary Clinton	Neutro
“Jeb Bush é o único candidato são entre os republicanos”	Donald Trump	Contrário

Fonte: MOHAMMAD et al. 2016 - Adaptado

A detecção automática de posicionamento possui aplicações diversas nas áreas de Recuperação de Informação, sumarização de texto, análise de sentimentos e outros (MOHAMMAD et al., 2016). Na última década, as pesquisas relacionadas à detecção de posicionamento se concentraram em debates (congressuais ou em fóruns online) (MOHAMMAD et al. 2016) e em textos de mídias sociais, como o *Twitter*.

Mais recentemente, estas últimas também começaram a ser utilizadas para a divulgação de notícias em tempo real, para uma audiência global. Tal conjuntura permitiu a disseminação de notícias falsas e rumores sem verificação. Surgiram então organizações de verificadores de fatos, que realizam esta tarefa manualmente. Recentemente, no entanto, foram lançadas várias ferramentas que procuram automatizar este processo, como o *Fact Check* (GOOGLE, 2017) e até uma plataforma de notícias com verificação comunitária, o *Wikitribune*, lançado pelo fundador da *Wikipedia*, Jimmy Wales (INDEPENDENT, 2017).

Como a tarefa de identificação de notícias falsas é muito complexa para os métodos de Processamento de Linguagem Natural atuais, a detecção de posicionamento apresentou-se como um primeiro passo na construção de tal sistema (FERREIRA; VLACHOS; 2016). Neste contexto definiu-se a primeira edição do *Fake News Challenge*, uma competição internacional voltada a fomentar o desenvolvimento de ferramentas de verificação de notícias.

2.1 Trabalhos Recentes

Thomas et al. (2006) criaram um *corpus* de transcrições dos debates realizados no Congresso americano no ano de 2005, além dos registros de votos para todas as votações daquele ano. Eles então utilizaram um classificador do tipo de máquina de vetor suporte (daqui em diante denotado por SVM, do inglês *Support Vector Machine*) para identificar o posicionamento de um discurso em relação a uma proposta de lei, utilizando os registros de votações como *ground truth*. O trabalho utilizou ligações ponderadas entre os discursos relacionados, considerando o peso de uma ligação como o grau de probabilidade de que dois discursos recebam a mesma classificação. O melhor resultado obtido no conjunto de testes foi de 70, 81%.

Uma temática similar foi utilizada por Sridhar et al. (2014), aplicando a detecção de posicionamento a postagens feitas em fóruns *online*. O conjunto de dados utilizado foi obtido do *Internet Argument Corpus*, uma coleção anotada de 109.533 postagens. O sistema deveria identificar, dado um par postagem-resposta e um tópico, se a resposta seria favorável ou contrária ao tópico. Um conjunto de SVMs lineares foi treinado (cada uma especializada em um tópico). Foram utilizadas características linguísticas (*bag of words*, tamanho do texto, pontuações repetidas, quantidade de categorias léxicas), características dos autores do texto extraídas a partir de metadados e predicados lógicos. Foi obtido um F1 score de 0,74, com desvio padrão de 0,04.

Com a emergência do *Twitter* como meio de expressão de opiniões sobre tópicos diversos, vários trabalhos foram realizados utilizando *datasets* gerados a partir da coleta de *Tweets*. Mohammad et al. (2016) lançou um desafio chamado *SemEval-2016 Stance Detection for Twitter*, consistindo de duas sub-tarefas (A e B). Este desafio foi investigado por diversos grupos independentes. A tarefa A consistia em detectar o posicionamento de *tweets* em relação aos seguintes alvos: “Mudanças climáticas são uma preocupação verdadeira”, “Movimento feminista”, “Ateísmo”, “Legalização do aborto” e “Hillary Clinton”. Na tarefa B, o objetivo era determinar o posicionamento com relação a um alvo implícito, “Donald Trump”, com dados sem classificação prévia (AUGENSTEIN et al., 2016).

Foram implementadas versões *baseline* para ambas as tarefas, usando SVMs (em versões especializadas e um combinado), por Mohammad et al. (2016). Foram submetidos 19 classificadores para a tarefa A, com o maior F1 score de 0,6782, abaixo de um dos classificadores *baseline*. Já para a tarefa B, foram submetidos apenas 9 classificadores, com o melhor desempenho de 0,5628 (MOHAMMAD et al., 2016). Entre os participantes, destaca-se a abordagem utilizada por Wojatzki et al. (2016), que utilizou uma estratégia de classificadores complementares, numa abordagem similar a utilizada neste projeto, que será descrita no capítulo 4.

3. Técnicas Utilizadas

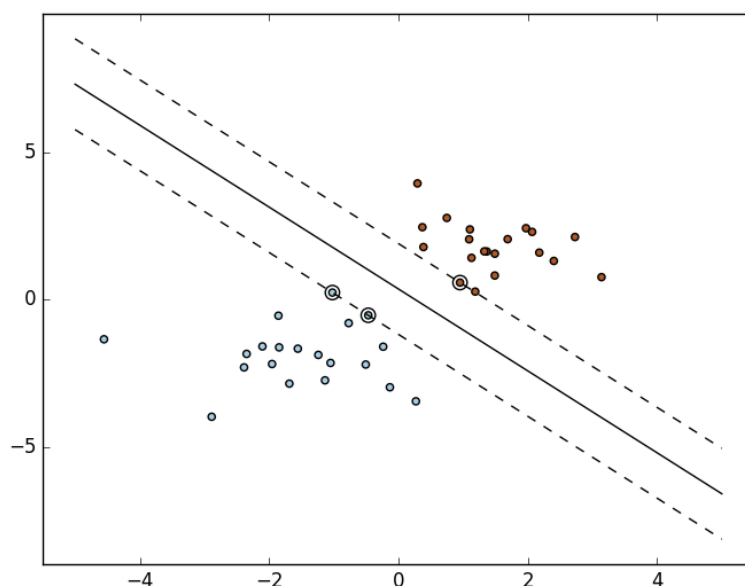
3.1 Classificadores

Foram testadas as variações de três tipos de classificadores: máquinas de vetores suporte, *Gradient Boosting*, e *Random Forest*. Todos os classificadores mencionados foram utilizados a partir da implementação em *Python* da biblioteca *scikit-learn*, a mais utilizada na área.

3.1.1 Máquinas de Vetores Suporte (SVMs)

SVMs são modelos de aprendizagem supervisionada que constroem um conjunto de hiperplanos num espaço de dimensionalidade alta, de maneira a classificar elementos em diferentes classes. Uma boa separação é obtida pelo hiperplano com a maior distância para os pontos mais próximos de qualquer classe (as margens), como visto na figura 1 (SCIKIT-LEARN, 2017).

Figura 1: Representação do Hiperplano gerado por um SVM e suas margens



Fonte: SCIKIT-LEARN, 2017

Os classificadores SVM utilizam-se de uma função de *kernel* para mapear os vetores de características para o espaço onde o hiperplano está definido. Neste experimento foi utilizada a versão do SVM com *kernel* linear, o *Linear Support Vector Classification* (*LinearSVC*). A figura 2 apresenta os parâmetros possíveis para utilização do *LinearSVC*, bem como um exemplo de utilização.

Figura 2: Exemplo de implementação do *LinearSVC*

```
1  """
2  class sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001,
3  C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None,
4  verbose=0, random_state=None, max_iter=1000)
5  """
6  from sklearn import svm
7
8  X = [[0, 0], [1, 1]]
9  y = [0, 1]
10
11  clf = svm.LinearSVC()
12
13  # Training
14  clf.fit(X, y)
15
16  # Predicting new values
17  print(clf.predict([[2., 2.]])
```

Fonte: SCIKIT-LEARN, 2017 - Adaptado

Classificadores SVM são normalmente efetivos em espaços de alto número de dimensões, além de serem eficientes em termos de memória e versáteis, devido a diferentes funções *kernel*. No entanto, o processo de treinamento e classificação é muito custoso, e a grande quantidade de parâmetros torna proibitivo o uso de técnicas de *fine-tuning* para conjuntos de dados muito grandes, como o apresentado neste projeto (e descrito em detalhes no capítulo 4).

3.1.2 Classificador *Gradient Tree Boosting*

Em alguns casos, é mais vantajoso combinar o resultado de vários classificadores bases em vez de utilizar apenas um. Classificadores mais complexos foram construídos desta maneira, do qual o *Gradient Tree Boosting* é um exemplo. Nele, estimadores base (do tipo árvore de decisão) são construídos sequencialmente, e um deles tenta reduzir o viés do classificador combinado, mesclando vários modelos fracos em um conjunto poderoso (SCIKIT-LEARN, 2017).

Suas vantagens incluem robustez a elementos *outliers* e dados com características heterogêneas. No entanto, como o *boosting* é sequencial, não pode ser paralelizado, o que gera um custo alto de treinamento e classificação. A implementação utilizada neste projeto foi o *GradientBoostingClassifier*, que suporta tanto classificações binárias como multiclases. É possível controlar o número de estimadores, bem como o tamanho de cada árvore. A figura 3 apresenta todos os possíveis parâmetros de utilização, bem como um exemplo de uso.

Figura 3: Exemplo de implementação do *GradientBoostingClassifier*

```
1  """
2  class sklearn.ensemble.GradientBoostingClassifier(loss='deviance', learning_rate=0.1, n_estimators=100,
3  subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1,
4  min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_split=1e-07, init=None, random_state=None,
5  max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto')
6  """
7  from sklearn.datasets import make_hastie_10_2
8  from sklearn.ensemble import GradientBoostingClassifier
9
10 X, y = make_hastie_10_2(random_state=0)
11 X_train, X_test = X[:2000], X[2000:]
12 y_train, y_test = y[:2000], y[2000:]
13
14 clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
15 ...     max_depth=1, random_state=0).fit(X_train, y_train)
16 print(clf.score(X_test, y_test))
```

Fonte SCIKIT-LEARN, 2017

3.1.3 Classificador *Random Forest*

A motivação do classificador *Random Forest* é similar a do *Gradient Tree Boosting*. No entanto, em vez de construir estimadores sequencialmente, estes são construídos de maneira independente, e o resultado final é a média dos resultados individuais. No caso específico do *Random Forest*, um conjunto de árvores de decisão randômico é gerado através do conjunto de treinamento, através da escolha randômica de separações, ao invés da melhor separação possível. O resultado da predição final é a média das previsões das árvores individuais. Como resultado da aleatoriedade, a variância do modelo diminui, gerando um melhor resultado. Como cada árvore é independente, este classificador pode ser paralelizado, o que reduz de maneira considerável o custo de treinamento e classificação. A implementação utilizada neste projeto foi o *RandomForestClassifier*, da biblioteca *scikit-learn*. A figura 4 apresenta todos os possíveis parâmetros de utilização, bem como um exemplo de uso.

Figura 4: Exemplo de implementação do *RandomForestClassifier*

```
1  """
2  class sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini',
3  max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
4  max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07, bootstrap=True,
5  oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None)
6  """
7  from sklearn.ensemble import RandomForestClassifier
8  X = [[0, 0], [1, 1]]
9  Y = [0, 1]
10 clf = RandomForestClassifier(n_estimators=10)
11 print(clf = clf.fit(X, Y))
```

Fonte: SCIKIT-LEARN, 2017

3.2 Características

Todos os classificadores mencionados anteriormente recebem uma representação, tanto do conjunto de treinamento como de teste, no formato de um vetor de características, ou *features*. Para o experimento, foram implementadas 9 *features* diferentes, além das 4 previamente fornecidas pela competição. O vetor de características final gerado possuía 1568 dimensões. Como o problema envolvia estabelecer a relação entre um texto e outro, algumas das *features* escolhidas envolvem comparações entre os dois textos. Além disso, o tamanho do conjunto de dados resultou na utilização de quantificações e reduções de características mais extensas, para chegar a um vetor de características que fosse computacionalmente operável. A implementação das *features* está disponível no Apêndice A, ao final deste trabalho.

3.2.1 Interseção de palavras

A interseção de palavras é uma característica que utiliza os dois textos providos (a manchete e o corpo da notícia), para gerar um vetor unitário composto da normalização da quantidade de palavras em comum entre os dois textos, de acordo com a equação 1. Antes de realizar este cálculo, cada texto é limpo, com a retirada de caracteres não alfanuméricos e conversão para minúsculas. Na equação 1, M corresponde ao conjunto de palavras contidas na manchete, enquanto C é o conjunto de palavras contidas no corpo da notícia. Esta *feature* foi fornecida com o código base da competição.

$$\frac{|M \cap C|}{|M \cup C|}$$

Equação 1: Cálculo da interseção de palavras

3.2.2 Palavras de refutação

Como a discordância entre manchete e corpo de notícia poderia ser identificada através da presença de palavras com sentido de negação, esta característica também foi incluída no experimento. Foi utilizada a implementação fornecida pela competição. Como resultado, foi gerado um vetor binário para cada par, com o valor 1 indicando a presença da palavra correspondente no Quadro 2 no texto da manchete, e 0 a sua ausência.

Quadro 2: Palavras de refutação

“fake”, “fraud”, “hoax”, “false”, “deny”, “denies”, “not”, “despite”, “nope”, “doubt”, “doubts”, “bogus”, “debunk”, “pranks”, “retract”.
--

Fonte: Autora

3.2.3 Polaridade

As mesmas palavras do Quadro 2 também foram utilizadas para calcular a polaridade da manchete e do corpo da notícia. Neste caso, utilizou-se a hipótese que duas palavras de refutação juntas se anulam. Assim, após a limpeza e a *tokenization* (processo de repartir um texto em *tokens*, unidades de significado mínimo) dos textos, era retornado um vetor com dois elementos, a polaridade da manchete e a do corpo. Um texto com um número par de palavras de refutação possui polaridade zero, enquanto um texto com número ímpar possui polaridade 1.

3.2.4 Co-ocorrência

Esta *feature* também indica a relação entre os dois textos fornecidos. Ela conta a quantidade de vezes que um *token*, *n-gram* ou *char-gram* da manchete aparece no corpo de texto, concatenando-as num único vetor. *N-grams* correspondem a um conjunto de tamanho N de *tokens*, enquanto *char-grams* correspondem a um conjunto de tamanho M de caracteres. Para o experimento, foram utilizados $M = \{2, 4, 8, 16\}$ para *chargrams* e $N = \{2, 3, 4, 5, 6\}$ para *n-grams*. Assim como as features anteriores, esta também foi fornecida com o código base da competição.

3.2.5 Bag of Words

Esta *feature* clássica do Processamento de Linguagem Natural consiste em, a partir de um vocabulário dado, transformar um texto em um vetor de inteiros, onde cada valor corresponde à quantidade de vezes que cada palavra deste vocabulário aparece no texto. Neste projeto foi utilizada a implementação fornecida pela biblioteca *scikit-learn*, a *CountVectorizer*. Como o vocabulário precisa ser global, esta feature é calculada sobre

todo o conjunto de treinamento de uma só vez. Por limitações de memória e processamento, devido ao grande conjunto de dados, só foram selecionadas as 200 palavras mais frequentes em todo o conjunto de dados de treinamento.

3.2.6 *Term Frequency – Inverse Document Frequency (TF-IDF)*

O TF-IDF de um conjunto de documentos corresponde à matriz formada de acordo com as equações 2 e 3. Nelas, $tf(t, d)$ corresponde à quantidade de vezes que o termo t aparece no documento d , enquanto que n_d é o número total de documentos e $df(d, t)$ é a quantidade de documentos que contém o termo t . Esta transformação provê uma maneira de identificar os termos mais relevantes na diferenciação de um documento de outro, já que termos que aparecem em todos os documentos não possuem grande informação sobre os mesmos. Assim como na *bag of words*, foi utilizada a implementação fornecida pela biblioteca *scikit-learn*, através da classe *TfidfVectorizer*, também limitada aos 200 termos mais relevantes.

$$tfidf(d, t) = tf(t, d) \times idf(t)$$

Equação 2: Cálculo do TF-IDF

$$idf(t) = \log \frac{n_d}{1 + df(d, t)}, n_d \rightarrow \text{número de documentos}$$

Equação 3: Cálculo do IDF

3.2.7 **Cosseno**

Dadas as representações de dois documentos em vetores de *bag of words*, o cosseno entre estes dois elementos é definido como o produto interno entre estes dois vetores, normalizado por seus módulos. Trata-se então de uma medida de similaridade, aplicada entre as representações geradas previamente pelo *bag of words* para cada par manchete-notícia. Foi utilizada a implementação fornecida pela *scikit-learn*, a função *cosine_similarity*.

3.2.8 *Part-of-speech Tagging*

O processo de *Part-of-speech (POS) tagging* corresponde a categorizar *tokens* de um documento através de definições gramaticais, como substantivo, adjetivo, verbo, etc. Assim, podemos associar as quantidades de cada tipo de partícula gramatical de maneira a gerar características para o conjunto de dados. Neste experimento foram geradas duas *features* utilizando-se da biblioteca *spaCy*, voltada especificamente para Processamento de Linguagem Natural em *Python*. A primeira *feature* gerou um “*bag of tags*”, com as quantidades de cada texto em relação as *tags* identificadas no vocabulário. A segunda característica obtida foi o cálculo do cosseno entre os vetores de “*bag of tags*” de cada par manchete-notícia. Essa abordagem visou tanto reduzir o tamanho do vetor de características como utilizar a informação contida na frequência de termos verbais. Por exemplo, frases com muitos adjetivos podem indicar uma opinião, enquanto que textos com uma quantidade considerável de verbos possuem um perfil mais descritivo.

3.2.9 Extração de tópicos

A extração de tópicos consiste de gerar um conjunto de tópicos semânticos de um documento a partir de grupos de *tokens*. Neste experimento foi utilizada a implementação do algoritmo *Latent Dirichlet allocation* fornecida pela *scikit-learn*. Este algoritmo considera que documentos cobrem apenas um pequeno conjunto de tópicos, que por sua vez usam um pequeno conjunto de palavras frequentemente, gerando uma atribuição mais precisa de tópicos para um texto. Foram calculadas duas *features* a partir deste conceito: os tópicos classificados diretamente pelo algoritmo, e o cosseno dos vetores de tópicos de um par manchete-notícia. O intuito desta *feature* foi determinar manchetes e corpos de texto relacionados ou não, de acordo com seu conjunto de tópicos.

3.2.10 Proporção de antônimos

A quantidade de antônimos presentes no corpo da notícia em relação à manchete é um bom indicador da relação entre eles. Assim, a partir da obtenção do conjunto de antônimos de todas as palavras da manchete (utilizando a biblioteca *scikit-learn* para obter acesso a

WordNet, o conjunto de dados léxicos da língua inglesa), foi computado, para o corpo correspondente, a proporção de antônimos presentes em comparação com o tamanho do corpo.

3.2.11 *Word2vec*

Este modelo, desenvolvido por Mikolov et al. (2013) é utilizado para aprender representações vetoriais de palavras, os chamados “*word embeddings*”. O objetivo principal deste modelo é representar palavras num espaço vetorial contínuo onde elementos similares semanticamente sejam mapeados para pontos próximos entre si. O modelo *Word2vec* prediz palavras a partir de suas vizinhas, a partir de vetores de *embeddings* pequenos e densos, aprendidos previamente. Foi utilizada a implementação da biblioteca *spaCy*, que calcula automaticamente os *word embeddings* para cada termo de cada documento.

4. Fake News Challenge

A competição propriamente dita do *Fake News Challenge* ocorreu entre dezembro de 2016, quando o cadastramento foi aberto, e 15 de junho, quando o resultado final foi anunciado. Foram disponibilizados aos participantes um conjunto de dados classificado para ser utilizado como única forma de treinamento possível, além de um sistema *baseline* composto por um classificador simples do tipo *GradientBoostingClassifier*, com um conjunto de quatro *features* implementadas (interseção de palavras, refutação, polaridade e co-ocorrência).

O conjunto de dados de treinamento e o de teste, fornecido 2 dias antes do fim do prazo de submissões (3 de junho de 2017), são formados por linhas que relacionam um corpo de notícia a uma manchete, não necessariamente relacionados. O conjunto de treinamento possui também as informações sobre os diferentes posicionamentos a serem classificados:

- *unrelated*: a manchete e o corpo da notícia tratam de assuntos diferentes;
- *discuss*: o corpo da notícia não apresenta juízo de valor sobre a manchete, mas é relacionado com ela;
- *agree*: o corpo da notícia confirma o conteúdo da manchete;
- *disagree*: o corpo da notícia refuta o conteúdo da manchete.

O conjunto de dados de treinamento consiste de aproximadamente 50000 pares manchete-notícia, categorizados entre as quatro classes acima. A distribuição entre as classes é extremamente desigual, como podemos ver no Quadro 3.

Quadro 3: Distribuição das classes no conjunto de treinamento

Instâncias	<i>unrelated</i>	<i>discuss</i>	<i>agree</i>	<i>disagree</i>
49972	73,131%	17,828%	7,36012%	1,68094%

Fonte: FNC-1, 2017

Além disso, os pares foram gerados de maneira aleatória entre um conjunto determinado de manchetes e notícias. Assim, é impossível separar o conjunto fornecido em treinamento e teste sem que haja repetição de notícias ou manchetes em ambos (MROWCA et al., 2017). Assim, existe uma considerável variação entre os resultados obtidos no conjunto de treinamento, explicitados neste trabalho, e os obtidos quando aplicados ao conjunto de teste fornecido posteriormente. Como, até o momento da

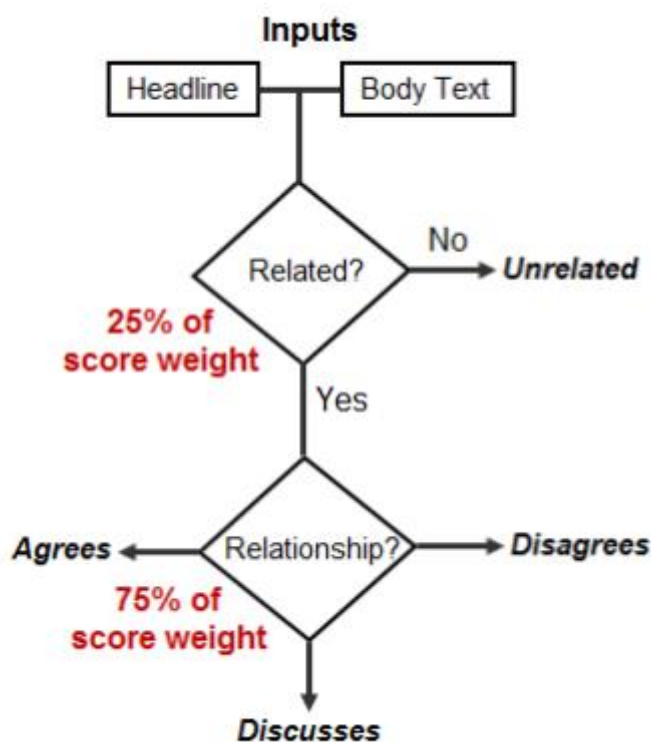
finalização deste relatório, o conjunto de teste anotado não foi disponibilizado, não há como ter acesso a métricas de performance para este conjunto.

A avaliação dos classificadores submetidos na competição não leva em consideração diretamente as métricas de precisão e acurácia de cada classe. Ao invés disso, são determinados dois níveis de pontuação, de acordo com a seguinte classificação:

- **Nível 1:** Classificar o par manchete-notícia entre *unrelated* ou *related*;
- **Nível 2:** Classificar pares do tipo *related* em *agree*, *disagree* ou *discuss*.

Como a primeira parte é considerada mais fácil, a classificação do nível 1 vale apenas 25% da pontuação final, enquanto que a segunda parte é responsável por 75% da pontuação, como indicado na Figura 5.

Figura 5: Sistema de classificação do *Fake News Challenge*



Fonte: FAKE NEWS CHALLENGE, 2016

A pontuação final é então calculada a partir desta distribuição. Foi fornecido um *script Python* que calcula automaticamente a pontuação final a partir do conjunto de pares classificados e pela classificação verdadeira. Este script gera uma matriz de confusão, bem como uma pontuação absoluta entre 0 e 4448.5, a pontuação máxima, obtida por um classificador hipotético que acertasse todas as classificações do conjunto de treinamento.

O classificador *baseline* fornecido obtém uma pontuação de 3538 pontos, ou seja, 79.53% do total. Podemos ver sua tabela de distribuição no Quadro 4.

Quadro 4: Resultados do classificador *baseline* fornecido

	<i>AGREE</i>	<i>DISAGREE</i>	<i>DISCUSS</i>	<i>UNRELATED</i>
<i>AGREE</i>	118	3	556	85
<i>DISAGREE</i>	14	3	130	15
<i>DISCUSS</i>	58	5	1527	210
<i>UNRELATED</i>	5	1	98	6794
<i>SCORE</i>	3538 / 4448,5 ou 79,53%			

Fonte: FNC-1-BASELINE, 2017

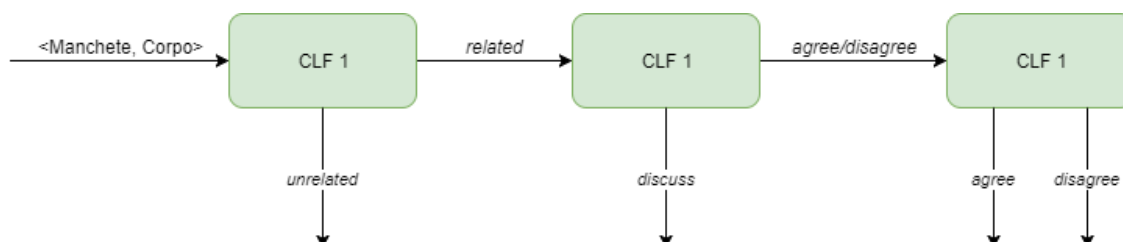
4.1 Arquitetura

A fim de otimizar ao máximo a classificação e manter a separação semântica entre as diferentes classes do problema, foi idealizada uma arquitetura composta por três classificadores independentes, cada um responsável por uma classificação binária, delimitada a seguir:

- **Classificador 1:** Classifica os pares manchete-notícia em *unrelated* e *related*;
- **Classificador 2:** Classifica os pares *related* entre *discuss* e *agree/disagree*;
- **Classificador 3:** Classifica os pares restantes em *agree* e *disagree*.

Assim, cada classificador só é treinado com os pares relevantes para seu problema específico. Inicialmente pretendia-se também separar as *features* que cada um utilizava, mas não houve tempo hábil para implementar esta estratégia de maneira satisfatória. Da mesma maneira, cada classificador só classificará os pares que chegarem até a ele através da classificação anterior na sequência. A figura 6 ilustra a arquitetura implementada.

Figura 6: Arquitetura de classificadores implementada



Fonte: Autora

4.2 Submissão

O *script* final gerado possibilitava ao usuário escolher um entre três tipos de classificadores diferentes (SVM, *Gradient Boosting* ou *Random Forest*), além do nome do arquivo no qual o classificador seria salvo para posterior utilização no conjunto de teste. O usuário determina também a porcentagem do conjunto de treinamento a ser utilizado como *holdout* para avaliação dos classificadores (para a competição, os classificadores foram treinados sobre todo o conjunto de treinamento). Um outro *script* recupera, então, um classificador salvo a partir do comando do usuário e o executa sobre arquivos também determinados pelo usuário, para enfim gerar um arquivo de extensão *csv* com os pares classificados, utilizado no sistema de submissão adotado pela competição. Ambos os scripts estão disponíveis nos Apêndices B e C, respectivamente.

Foram permitidas até 5 submissões por cada equipe, com o resultado de cada submissão sendo visível, e a submissão final sendo a considerada para fins de classificação.

5. Resultados

Durante a competição, os melhores resultados alcançados foram com uma combinação de três classificadores do tipo *Gradient Boosting*. A equipe do Centro de Informática terminou em 12º lugar. Cerca de 50 equipes realizaram ao menos algum tipo de submissão.

5.1 Conjunto de Treinamento

Foram realizados testes com os três tipos de classificadores diferentes, utilizando 20% do conjunto de treinamento como *holdout* para avaliação de cada classificador. Os resultados foram condizentes (embora com uma maior acurácia, devido aos problemas relatados no capítulo 4) com os observados durante a competição propriamente dita. O classificador *Gradient Boosting* obteve o melhor desempenho, com o *Random Forest* próximo, porém inferior, e o SVM com uma pontuação mais baixa. É importante ressaltar que essas pontuações não correspondem a acurácia total e sim aos pesos discutidos no capítulo 4. Os resultados obtidos para cada classificador encontram-se nos Quadros 5, 6 e 7 abaixo.

Quadro 5: Resultados da arquitetura baseada em classificadores *Gradient Boosting*

Real	Previsão				
		<i>AGREE</i>	<i>DISAGREE</i>	<i>DISCUSS</i>	<i>UNRELATED</i>
	<i>AGREE</i>	453	11	284	14
	<i>DISAGREE</i>	57	55	48	2
	<i>DISCUSS</i>	164	18	1588	30
	<i>UNRELATED</i>	49	2	117	6730
	<i>SCORE</i>	3924 / 4448,5 ou 88,21%			

Fonte: Autora

Quadro 6: Resultados da arquitetura baseada em classificadores *Random Forest*

Real	Previsão				
	<i>AGREE</i>	<i>DISAGREE</i>	<i>DISCUSS</i>	<i>UNRELATED</i>	
	<i>AGREE</i>	430	6	300	26
	<i>DISAGREE</i>	84	31	45	2
	<i>DISCUSS</i>	157	13	1586	44
	<i>UNRELATED</i>	13	0	92	6793
	<i>SCORE</i>	3896,5 / 4448,5 ou 87,59%			

Fonte: Autora

Quadro 7: Resultados da arquitetura baseada em classificadores *SVM*

		Previsão			
		<i>AGREE</i>	<i>DISAGREE</i>	<i>DISCUSS</i>	<i>UNRELATED</i>
Real	<i>AGREE</i>	464	49	209	40
	<i>DISAGREE</i>	55	52	48	7
	<i>DISCUSS</i>	295	61	1326	118
	<i>UNRELATED</i>	135	30	217	6516
	<i>SCORE</i>	3650,25 / 4448,5 ou 82,06%			

Fonte: Autora

5.2 Conjunto de Testes

O conjunto de testes divulgado possuía 25413 instâncias, com a distribuição definida no Quadro 8, abaixo:

Quadro 8: Distribuição das classes no conjunto de treinamento

Instâncias	<i>unrelated</i>	<i>discuss</i>	<i>agree</i>	<i>disagree</i>
24513	72,203%	17,566%	7,488%	2,743%

Fonte: FNC-1, 2017

Com os classificadores treinados em 100% do conjunto de treinamento, os resultados obtidos no conjunto de teste encontram-se nos Quadros 9, 10 e 11 abaixo.

Quadro 9: Resultados da arquitetura baseada em classificadores *Gradient Boosting*

		Previsão			
		<i>AGREE</i>	<i>DISAGREE</i>	<i>DISCUSS</i>	<i>UNRELATED</i>
Real	<i>AGREE</i>	1073	13	584	233
	<i>DISAGREE</i>	290	17	193	197
	<i>DISCUSS</i>	831	37	3166	430
	<i>UNRELATED</i>	226	7	400	17716
	<i>SCORE</i>	9172 / 11651,25 ou 78,72%			

Fonte: Autora

Quadro 10: Resultados da arquitetura baseada em classificadores *Random Forest*

		Previsão			
		<i>AGREE</i>	<i>DISAGREE</i>	<i>DISCUSS</i>	<i>UNRELATED</i>
Real	<i>AGREE</i>	939	0	713	251
	<i>DISAGREE</i>	255	0	220	222
	<i>DISCUSS</i>	736	0	3165	563
	<i>UNRELATED</i>	86	0	289	17974
	<i>SCORE</i>	9078,5 / 11651,25 ou 77,92%			

Fonte: Autora

Quadro 11: Resultados da arquitetura baseada em classificadores *SVM*

Real	Previsão				
	<i>AGREE</i>	<i>DISAGREE</i>	<i>DISCUSS</i>	<i>UNRELATED</i>	
	<i>AGREE</i>	856	62	682	303
	<i>DISAGREE</i>	167	33	237	260
	<i>DISCUSS</i>	829	111	2729	795
	<i>UNRELATED</i>	114	50	396	17789
	<i>SCORE</i>	8587,25 / 11651,25 ou 73,70%			

Fonte: Autora

Os dados obtidos retratam a dificuldade de classificar corretamente os pares da classe *disagree* por todos os classificadores. A pouca quantidade de pares desta classe no conjunto de treinamento (cerca de 1,5% do total) certamente contribuiu para este resultado. No entanto, o conjunto de *features* e a arquitetura sequencial dos classificadores contribuiu para aumentar de maneira considerável o desempenho em relação ao classificador *baseline*, e o desempenho na competição final credencia este trabalho a uma considerável expansão futura, a ser tratada no próximo capítulo. O melhor classificador para o conjunto de teste foi o *Gradient Tree Boosting*, com um desempenho ligeiramente melhor em relação ao *Random Forest*. Ambos são do tipo *ensemble*, que usam combinações de resultados de estimadores individuais para definir a classe de um elemento. O desempenho no conjunto de testes caiu em relação ao *holdout* de treinamento. O classificador SVM, originalmente pensado como a melhor solução, mostrou ter o pior desempenho em todas as medições.

O desempenho dos classificadores pode ser melhorado utilizando-se de técnicas de amostragem para aumentar a proporção dos elementos da classe *disagree*, por exemplo, ou utilizando mais características relativas à polaridade interna dos pares, já que a classificação inicial entre *related* e *unrelated* apresentou um excelente desempenho em todas as configurações testadas.

6. Conclusão

O principal objetivo deste projeto foi construir um sistema de classificação de posicionamento, dado um par manchete-notícia, analisando diversas possibilidades de classificadores e características. Este sistema foi planejado como submissão de uma equipe do Centro de Informática no *Fake News Challenge*, a primeira competição de Detecção de Posicionamento aplicada ao contexto de verificação de notícias. Este assunto possui uma relevância atual imensa, devido à proliferação de organizações que se propõem a disseminar notícias falsas, utilizando-se de redes sociais como o *Facebook* e o *Twitter*.

Inicialmente foi definida a tarefa de Detecção de Posicionamento, bem como os outros conceitos básicos necessários ao entendimento do problema. Foram analisados trabalhos recentes realizados na área, que ainda é muito recente no contexto do Processamento de Linguagem Natural.

Em seguida, foram definidos os classificadores e características utilizados no projeto, com suas adaptações realizadas no contexto deste projeto. Foram utilizados três tipos de classificadores diferentes: máquinas de vetores suporte (SVMs), classificadores do tipo *Gradient Boosting* e *Random Forest*. Também foram implementadas 13 *features* diferentes no total, todas detalhadas neste relatório.

A competição objetivo deste trabalho também foi descrita, com suas definições de conjuntos de dados de treino e teste, pontuação e prazos. Foi utilizada no sistema uma arquitetura sequencial de três classificadores, para separar os pares de manchete-notícia em quatro classes distintas: *unrelated*, *discuss*, *agree* e *disagree*. Foram também detalhados os *scripts* desenvolvidos para teste e submissão final da competição.

O sistema proposto e implementado atingiu resultados satisfatórios, com a equipe do Centro de Informática alcançando o 12º lugar. Foram analisados então resultados sobre o conjunto de treinamento, com um *holdout* de 20%, e sobre o conjunto de testes fornecido pela competição. Os melhores resultados obtidos foram com três classificadores *GradiengBoosting*. Considerando a melhora obtida em relação ao classificador *baseline* fornecido, o sistema se mostrou promissor.

Em geral, este projeto conseguiu cumprir de maneira satisfatória seu objetivo principal. Ele também se mostrou bastante extensível para diversas aplicações no futuro, tratadas na seção seguinte.

6.1 Trabalhos Futuros

Como esta foi a primeira edição do *Fake News Challenge*, extensões naturais deste trabalho consistem de edições futuras desta competição, que certamente irão tratar de outras tarefas relacionadas à verificação de veracidade de notícias. Uma combinação diferente de características por classificador também poderá ser estudada.

Finalmente, um trabalho futuro relevante seria uma implementação semelhante desta ferramenta, utilizando o português como língua base. Trabalhos acadêmicos e até comerciais nesta área utilizando a língua portuguesa são inexistentes, e os desafios são muito maiores. Uma ferramenta deste tipo, no entanto, poderia ser utilizada por veículos jornalísticos de maneira a melhorar a qualidade das notícias difundidas, por exemplo.

Apêndice A – Implementação das *features* do sistema

Figura 7: Implementação das *features* utilizadas no sistema

```
1 import os
2 import re
3 import nltk
4 import spacy
5 import numpy as np
6 from sklearn import feature_extraction
7 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
8 from sklearn.decomposition import LatentDirichletAllocation
9 from sklearn.metrics.pairwise import cosine_similarity
10 from tqdm import tqdm
11 import pickle
12
13 _wnl = nltk.WordNetLemmatizer()
14
15 def normalize_word(w):
16     return _wnl.lemmatize(w).lower()
17
18 nlp_fast = spacy.load('en', parser=False, entity=False)
19 def get_tokenized_lemmas(s):
20     return [w.lemma_ for w in nlp_fast(s)]
21
22 def get_tokenized_tags(s):
23     return [w.tag_ for w in nlp_fast(s)]
24
25 def clean(s):
26     # Cleans a string: Lowercasing, trimming, removing non-alphanumeric
27     return " ".join(re.findall(r'\w+', s, flags=re.UNICODE)).lower()
28
29 def remove_stopwords(l):
30     # Removes stopwords from a list of tokens
31     return [w for w in l if w not in feature_extraction.text.ENGLISH_STOP_WORDS]
32
33 def gen_or_load_feats(feats_fn, headlines, bodies, split, feature_file):
34     if not os.path.isfile(feature_file):
35         feats = feats_fn(headlines, bodies, split)
36         np.save(feature_file, feats)
37
38     return np.load(feature_file)
39
40 def word_overlap_features(headlines, bodies, split='train'):
41     X = []
42     for i, (headline, body) in tqdm(enumerate(zip(headlines, bodies))):
43         clean_headline = clean(headline)
44         clean_body = clean(body)
45         clean_headline = get_tokenized_lemmas(clean_headline)
46         clean_body = get_tokenized_lemmas(clean_body)
47         features = [
48             len(set(clean_headline).intersection(clean_body)) / float(len(set(clean_headline).union(clean_body)))]
49         X.append(features)
50     return X
51
52 def refuting_features(headlines, bodies, split='train'):
53     _refuting_words = [
54         'fake',
55         'fraud',
56         'hoax',
57         'false',
58         'deny', 'denies',
59         'not',
60         'despite',
61         'nope',
62         'doubt', 'doubts',
63         'bogus',
64         'debunk',
65         'pranks',
66         'retract'
67     ]
68     X = []
69     for i, (headline, body) in tqdm(enumerate(zip(headlines, bodies))):
70         clean_headline = clean(headline)
71         clean_headline = get_tokenized_lemmas(clean_headline)
72         features = [1 if word in clean_headline else 0 for word in _refuting_words]
73         X.append(features)
74     return X
75
```

```

76 def polarity_features(headlines, bodies, split='train'):
77     _refuting_words = [
78         'fake',
79         'fraud',
80         'hoax',
81         'false',
82         'deny', 'denies',
83         'not',
84         'despite',
85         'nope',
86         'doubt', 'doubts',
87         'bogus',
88         'debunk',
89         'pranks',
90         'retract'
91     ]
92
93     def calculate_polarity(text):
94         tokens = get_tokenized_lemmas(text)
95         return sum([t in _refuting_words for t in tokens]) % 2
96
97     X = []
98     for i, (headline, body) in tqdm(enumerate(zip(headlines, bodies))):
99         clean_headline = clean(headline)
100        clean_body = clean(body)
101        features = []
102        features.append(calculate_polarity(clean_headline))
103        features.append(calculate_polarity(clean_body))
104        X.append(features)
105    return np.array(X)
106
107 def ngrams(input, n):
108     input = input.split(' ')
109     output = []
110     for i in range(len(input) - n + 1):
111         output.append(input[i:i + n])
112     return output
113
114 def chargrams(input, n):
115     output = []
116     for i in range(len(input) - n + 1):
117         output.append(input[i:i + n])
118     return output

```

```

119 def append_chargrams(features, text_headline, text_body, size):
120     grams = [' '.join(x) for x in chargrams(" ".join(remove_stopwords(text_headline.split()))), size]]
121     grams_hits = 0
122     grams_early_hits = 0
123     grams_first_hits = 0
124     for gram in grams:
125         if gram in text_body:
126             grams_hits += 1
127         if gram in text_body[:255]:
128             grams_early_hits += 1
129         if gram in text_body[:100]:
130             grams_first_hits += 1
131     features.append(grams_hits)
132     features.append(grams_early_hits)
133     features.append(grams_first_hits)
134     return features
135
136 def append_ngrams(features, text_headline, text_body, size):
137     grams = [' '.join(x) for x in ngrams(text_headline, size)]
138     grams_hits = 0
139     grams_early_hits = 0
140     for gram in grams:
141         if gram in text_body:
142             grams_hits += 1
143         if gram in text_body[:255]:
144             grams_early_hits += 1
145     features.append(grams_hits)
146     features.append(grams_early_hits)
147     return features
148
149 def hand_features(headlines, bodies, split='train'):
150
151     def binary_co_occurrence(headline, body):
152         # Count how many times a token in the title
153         # appears in the body text.
154         bin_count = 0
155         bin_count_early = 0
156         for headline_token in clean(headline).split(" "):
157             if headline_token in clean(body):
158                 bin_count += 1
159             if headline_token in clean(body)[:255]:
160                 bin_count_early += 1
161         return [bin_count, bin_count_early]
162

```

```

162
163 def binary_co_occurrence_stops(headline, body):
164     # Count how many times a token in the title
165     # appears in the body text. Stopwords in the title
166     # are ignored.
167     bin_count = 0
168     bin_count_early = 0
169     for headline_token in remove_stopwords(clean(headline).split(" ")):
170         if headline_token in clean(body):
171             bin_count += 1
172             bin_count_early += 1
173     return [bin_count, bin_count_early]
174
175 def count_grams(headline, body):
176     # Count how many times an n-gram of the title
177     # appears in the entire body, and intro paragraph
178
179     clean_body = clean(body)
180     clean_headline = clean(headline)
181     features = []
182     features = append_chargrams(features, clean_headline, clean_body, 2)
183     features = append_chargrams(features, clean_headline, clean_body, 8)
184     features = append_chargrams(features, clean_headline, clean_body, 4)
185     features = append_chargrams(features, clean_headline, clean_body, 16)
186     features = append_ngrams(features, clean_headline, clean_body, 2)
187     features = append_ngrams(features, clean_headline, clean_body, 3)
188     features = append_ngrams(features, clean_headline, clean_body, 4)
189     features = append_ngrams(features, clean_headline, clean_body, 5)
190     features = append_ngrams(features, clean_headline, clean_body, 6)
191     return features
192
193 X = []
194 for i, (headline, body) in tqdm(enumerate(zip(headlines, bodies))):
195     X.append(binary_co_occurrence(headline, body)
196             + binary_co_occurrence_stops(headline, body)
197             + count_grams(headline, body))
198
199 return X
200

```

```

200
201 def bag_of_words_fn(headlines, bodies, split='train'):
202     if not os.path.isfile('features/bow_transformer.pkl') and split=='train':
203         bow = CountVectorizer(analyzer='word', stop_words='english', max_features = 200)
204         bow.fit(bodies+headlines)
205         with open('features/bow_transformer.pkl', 'wb') as f:
206             pickle.dump(bow, f)
207     else:
208         print("Using precomputed bow transformer.")
209         bow = pickle.load(open('features/bow_transformer.pkl', 'rb'))
210         bodies_bow = bow.transform(bodies)
211         headlines_bow = bow.transform(headlines)
212         return np.c_[headlines_bow.toarray(), bodies_bow.toarray()]
213
214 def tf_idf_fn(headlines, bodies, split='train'):
215     if not os.path.isfile('features/tfidf_transformer.pkl') and split=='train':
216         tfidf = TfidfVectorizer(analyzer='word', stop_words='english', max_features = 200)
217         tfidf.fit(bodies+headlines)
218         with open('features/tfidf_transformer.pkl', 'wb') as f:
219             pickle.dump(tfidf, f)
220     else:
221         print("Using precomputed tfidf transformer.")
222         tfidf = pickle.load(open('features/tfidf_transformer.pkl', 'rb'))
223         headlines_tfidf = tfidf.transform(headlines)
224         bodies_tfidf = tfidf.transform(bodies)
225         return np.c_[headlines_tfidf.toarray(), bodies_tfidf.toarray()]
226
227 def cosine(headlines, bodies, split='train'):
228     if not os.path.isfile('features/bow_transformer.pkl') and split=='train':
229         bow = CountVectorizer(analyzer='word', stop_words='english', max_features = 200)
230         bow.fit(bodies+headlines)
231         with open('features/bow_transformer.pkl', 'wb') as f:
232             pickle.dump(bow, f)
233     else:
234         print("Using precomputed bow transformer.")
235         bow = pickle.load(open('features/bow_transformer.pkl', 'rb'))
236         bodies_bow = bow.transform(bodies)
237         headlines_bow = bow.transform(headlines)
238         X = []
239         for i, (headline, body) in tqdm(enumerate(zip(headlines_bow, bodies_bow))):
240             cos = cosine_similarity(headline, body)[0]
241             X.append(cos)
242         return np.array(X)
243
244 def word2vec(headlines, bodies, split='train'):
245     import spacy
246     nlp = spacy.load('en')
247     headline_vecs = [nlp(doc).vector for doc in tqdm(headlines)]
248     def max_sent_vec(body):
249         X = np.array([sent.vector for sent in nlp(body).sents])
250         return np.max(X, 0)
251     body_vecs = [max_sent_vec(body) for body in tqdm(bodies)]
252     return np.c_[headline_vecs, body_vecs]
253
254 def pos_tags(headlines, bodies, split='train'):
255     if not os.path.isfile('features/pos_transformer.pkl') and split=='train':
256         vec = CountVectorizer(analyzer=get_tokenized_tags)
257         vec.fit(bodies+headlines)
258         with open('features/pos_transformer.pkl', 'wb') as f:
259             pickle.dump(vec, f)
260     else:
261         print("Using precomputed POS transformer.")
262         vec = pickle.load(open('features/pos_transformer.pkl', 'rb'))
263         headlines_pos = vec.transform(headlines)
264         bodies_pos = vec.transform(bodies)
265
266         return np.c_[headlines_pos.toarray(), bodies_pos.toarray()]
267
268 def pos_tags_cosine(headlines, bodies, split='train'):
269     pos_features = np.load("features/pos.%.s.npy" % (split))
270     X = []
271     for arr in pos_features:
272         half = int(arr.size/2)
273         head = arr[:half]
274         body = arr[half:]
275         head = head.reshape(1, -1)
276         body = body.reshape(1, -1)
277         cos = cosine_similarity(head, body)[0]
278         X.append(cos)
279     return np.array(X)
280

```

```

280
281 def topic_features(headlines, bodies, split='train'):
282     if not os.path.isfile('features/topic_transformer.pkl') and split=='train':
283         vec = CountVectorizer(max_df=0.95, min_df=2, stop_words='english')
284         vec.fit(bodies+headlines)
285         with open('features/topic_transformer.pkl', 'wb') as f:
286             pickle.dump(vec, f)
287     else:
288         print("Using precomputed topic transformer.")
289         vec = pickle.load(open('features/topic_transformer.pkl', 'rb'))
290         bodies_transf = vec.transform(bodies)
291         headlines_transf = vec.transform(headlines)
292
293     if not os.path.isfile('features/lda.pkl') and split=='train':
294         lda = LatentDirichletAllocation(n_topics = 10, learning_method='online',
295                                         n_jobs = -1, random_state=1220)
296         lda.fit(bodies_transf+headlines_transf)
297         with open('features/lda.pkl', 'wb') as f:
298             pickle.dump(lda, f)
299     else:
300         print("Using precomputed LDA transformer.")
301         lda = pickle.load(open('features/lda.pkl', 'rb'))
302
303     bodies_topics = lda.transform(bodies_transf)
304     headlines_topics = lda.transform(headlines_transf)
305     return np.c_[headlines_topics, bodies_topics]
306
307 def topic_cosine(headlines, bodies, split='train'):
308     topic_features = np.load("features/topics.%s.npy" % (split))
309     X = []
310     for arr in topic_features:
311         half = int(arr.size/2)
312         head = arr[:half]
313         body = arr[half:]
314         head = head.reshape(1,-1)
315         body = body.reshape(1,-1)
316         cos = cosine_similarity(head, body)[0]
317         X.append(cos)
318     return np.array(X)
319
319
320 def antonym_rate(headlines, bodies, split='train'):
321     def get_synonyms_antonyms(s):
322         from nltk.corpus import wordnet
323         from itertools import chain
324         synonyms = []
325         antonyms = []
326         for syns in [wordnet.synsets(w.lower_) for w in nlp_fast(s)]:
327             for l in chain(*[syn.lemmas() for syn in syns]):
328                 synonyms.append(l.name())
329                 if l.antonyms():
330                     antonyms.append(l.antonyms()[0].name())
331         return set(synonyms), set(antonyms)
332
333     X = []
334     for i, (headline, body) in tqdm(enumerate(zip(headlines, bodies))):
335         _, h_antonyms = get_synonyms_antonyms(headline)
336         b_lower = set([w.lower_ for w in nlp_fast(body)])
337         rate = len(b_lower.intersection(h_antonyms)) / (len(h_antonyms)+1)
338         X.append(rate)
339     return np.array(X)

```

Fonte: Autora

Apêndice B – Script de treinamento

Figura 8: Implementação do script de treinamento

```
1 import sys
2 import pickle
3 import time
4 import numpy as np
5
6 # fnc
7 from utils.score import report_score, LABELS, score_submission
8 from utils.dataset import DataSet
9
10 # sklearn
11 from sklearn.metrics import classification_report
12 from sklearn.svm import LinearSVC
13 from sklearn.ensemble import RandomForestClassifier
14 from sklearn.ensemble import GradientBoostingClassifier
15 from sklearn.externals import joblib
16
17 # custom
18 from DataLoader import load_data, load_features
19 from pipelined_clf.PipelinedClassifier import PipelinedClassifier
20
21 if __name__ == "__main__":
22     classifiers = {'svm': LinearSVC(random_state=14128), 'rf': RandomForestClassifier(n_estimators=200, random_state=14128, n_jobs=-1),
23                  'gb': GradientBoostingClassifier(n_estimators=200, random_state=14128)}
24     classifier_names = {'svm': 'Support Vector Machine', 'rf': 'Random Forest Classifier',
25                        'gb': 'Gradient Boosting'}
26     BaseClassifier = classifiers['rf']
27     clf_name = classifier_names['rf']
28     clf_path = "clf.pkl"
29     training_pct = 1.0
30
31     if len(sys.argv) == 3:
32         BaseClassifier = classifiers[sys.argv[1]]
33         clf_name = classifier_names[sys.argv[1]]
34         clf_path = sys.argv[2]
35
36     print("Evaluating Pipelined Classifier: %s" % (clf_name))
37     start = time.time()
38     # train, folds and holdout are masks
39     df, train, folds, holdout = load_data(DataSet(), training=training_pct, n_folds=1)
40     print("Loading training features")
41     X_train, y_train, idx_arrs = load_features(df[train])
42     print("Loaded train data in " + str(time.time()-start) + " seconds")
43
44     clf = PipelinedClassifier(BaseClassifier, idx_arrs)
45     clf.fit(X_train, y_train)
46
47     with open(clf_path, 'wb') as f:
48         joblib.dump(clf, f)
49
50     if training_pct == 1:
51         print("Trained in 100% of set")
52         sys.exit(0)
53
54     print("Loading holdout features")
55     X_hold, y_hold, _ = load_features(df[holdout], 'holdout')
56
57     predicted = clf.predict(X_hold)
58     actual = y_hold
59
60     print("Test metrics")
61     print(classification_report(actual, predicted))
62     report_score(actual, predicted)
63     elapsed = time.time() - start
64     print("Elapsed time: %3.2f seconds" % (elapsed))
```

Fonte: Autora

Apêndice C – Script de submissão

Figura 9: Implementação do script de submissão

```
1  import sys
2  import time
3
4  # fnc
5  from utils.score import report_score
6
7  # sklearn
8  from sklearn.metrics import classification_report
9  from sklearn.externals import joblib
10
11 from csv import DictReader
12
13 # custom
14 from utils.dataset import DataSet
15 from pipelined_clf.PipelinedClassifier import PipelinedClassifier
16 from DataLoader import load_features, generate_dataframe
17 from SubmissionFormatter import SubmissionFormatter
18
19 if __name__ == "__main__":
20     bodies = "test_bodies.csv"
21     stances = "test_stances_unlabeled.csv"
22     clf_path = "rf_100.pkl"
23     if len(sys.argv) == 2:
24         clf_path = sys.argv[1]
25
26     clf = None
27     print("Loading saved classifier" )
28     with open(clf_path, "rb") as f:
29         clf = joblib.load(f)
30
31     if (clf is None):
32         print("Error loading classifier.")
33         sys.exit(1)
34
35     print("Loading test data")
36     # add test file paths here when available
37     dataset = DataSet(bodies=bodies, stances=stances)
38     df = generate_dataframe(dataset, split='test')
39
40     start = time.time()
41     print("Getting features for test data")
42     X_test = load_features(df, split='test')
43
44     print("Classifying test data")
45     predicted = clf.predict(X_test)
46     print(predicted)
47
```

```

47
48     clf2=None
49     with open("rf_100.pkl", "rb") as f:
50         clf2 = joblib.load(f)
51     predicted2 = clf2.predict(X_test)
52
53     disagree = 0
54     agree = 0
55     discuss = 0
56     pct=0
57     size=len(predicted2)
58     for i in range(size):
59         p2=predicted2[i]
60         p=predicted[i]
61         if p=='disagree':
62             disagree+=1
63         elif p=='agree':
64             agree+=1
65         elif p=='discuss':
66             discuss+=1
67
68         if p2==p:
69             pct+=1
70     unrelated = size-discuss-agree-disagree
71     print("Disagree count in sysarg clf: " + str(disagree))
72     print("Agree count in sysarg clf: " + str(agree))
73     print("Discuss count in sysarg clf: " + str(discuss))
74     print("Unrelated count in sysarg clf: " + str(unrelated))
75     print("Pct de iguais: " + str(pct/size))
76     elapsed = time.time() - start
77     print("Elapsed time: %3.2f seconds" % (elapsed))
78
79     # save result to csv
80     submission = SubmissionFormatter(df, predicted)
81     submission.save()

```

Fonte: Autora

Referências Bibliográficas

AUGENSTEIN, I.; ROCKTÄSCHEL, T.; VLACHOS, A.; BONTCHEVA, K.; "Stance Detection with Bidirectional Conditional Encoding", Proceedings of EMNLP 2016, 2016.

FACTMATA, "Introducing Factmata – Artificial Intelligence for automated fact-checking", 2016. Disponível em: <<https://medium.com/factmata/introducing-factmata-artificial-intelligence-for-political-fact-checking-db8acdbf4cf1>> [Acesso em 26/03/2017].

FAKE NEWS CHALLENGE, "Fake News Challenge", 2016. Disponível em: <<http://www.fakenewschallenge.org/>> [Acesso em 13/06/2017].

FERREIRA, W.; VLACHOS, A.; "Emergent: a novel data-set for stance classification", Proceedings of NAACL-HLT 2016, p. 1163-1168, 2016.

FNC-1, "Stance Detection dataset for FNC-1". Disponível em: <<https://github.com/FakeNewsChallenge/fnc-1>> [Acesso em 13/06/2017].

FNC-1-BASELINE - "Baseline FNC Implementation". Disponível em <<https://github.com/FakeNewsChallenge/fnc-1-baseline>> [Acesso em 13/06/2017].

GOOGLE, "Fact Check now available in Google Search and News around the world", 2017. Disponível em: <<https://blog.google/products/search/fact-check-now-available-google-search-and-news-around-world/>> [Acesso em 03/06/2017].

INDEPENDENT, "Wikipedia founder Jimmy Wales launching new site to tackle fake news", 2017. Disponível em: <[http://www.independent.co.uk/life-style/gadgets-and-](http://www.independent.co.uk/life-style/gadgets-and-news/)

tech/wikipedia-jimmy-wales-launching-site-combating-fake-news-wikitribune-fact-checking-a7700501.html> [Acesso em 03/06/2017].

MEISELMAN, Herbert L., "Emotion Measurement", 1. ed: Woodhead Publishing, 2016.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J.; "Efficient estimation of Word Representations in Vector Space", 2013.

MOHAMMAD, S. M.; KIRITCHENKO, S.; SOBHANI, P.; ZHU, X.; CHERRY, C.; "SemEval-2016 Task 6: Detecting Stance in Tweets", Proceedings of SemEval-2016, p. 31-41, 2016.

MROWCA, D.; WANG, E.; KOSSON, A.; "Stance Detection for Fake News Identification", 2017.

NEW YORK TIMES, "As Fake News spreads lies, more readers shrug at the Truth", 2016. Disponível em: <<https://www.nytimes.com/2016/12/06/us/fake-news-partisan-republican-democrat.html>> [Acesso em 21/03/2017].

PEW RESEARCH CENTER, "Many Americans believe Fake News is sowing confusion", 2016. Disponível em: <<http://www.journalism.org/2016/12/15/many-americans-believe-fake-news-is-sowing-confusion/>> [Acesso em 21/03/2017].

SCIKIT-LEARN, "1.4 Support Vector Machines". Disponível em: <<http://scikit-learn.org/stable/modules/svm.html>> [Acesso em 10/06/2017].

SHRIDAR, D.; GETOOR, L.; WALKER, M.; "Collective Stance Classification of Posts in Online Debate Forums", Proceedings of the Joint Workshop on Social Dynamics and Personal Attributes in Social Media, p. 109-177, 2014.

THOMAS, M; PANG, B.; LEE, L.; ". Get out the vote: Determining support or opposition from Congressional floor-debate transcripts.", Proceedings of EMNLP 2006, p. 327-335, 2006.

WOJATZKI, M.; ZESCH, T.; "Itl.uni-due at SemEval-2016 Task 6: Stance Detection in Social Media Using StackedClassifiers", Proceedings of SemEval-2016, p.428-433, 2016.