Classification of fake news article titles using character-level convolutional neural networks

Andrew Benecchi

2022-01-16

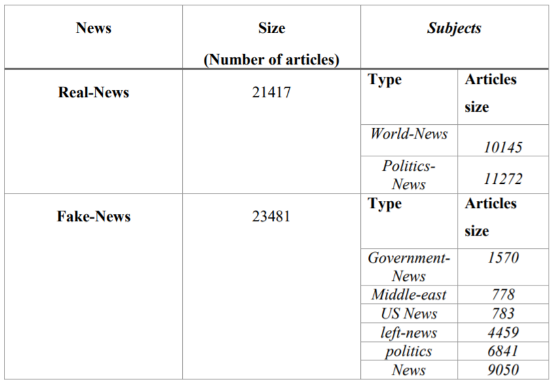
Abstract

Automated detection of spurious news and spam often rely on abstracting lexical, morphological, syntactical, or affective content from text. These models have merit in corresponding with human-observer features, but information is lost, and the features may not follow normal distributions, especially in short excerpts of text such as titles. Existing often rely on inclusion of the body text, which may or may not exist in real-world applications; since these article titles are accessible in metadata across platforms, classification of headlines has a much broader use case. We propose a method of applying character-level convolutional neural networks, a character-agnostic feature recognition system that combines the sentence-as-time-series philosophy of word2vec with a less constrained view of what information may be useful to classification, while vastly minimizing preprocessing time.

Introduction

Fake news has become pervasive within the public consciousness since the mass inter-generational adoption of multi-platform social networks such as Facebook, Twitter, and YouTube. The decentralized nature of social networks engenders the formation of echo chambers in online spaces, and growing distrust in government and mass media have not led to healthier skepticism towards all media but instead a growing trend for people to reject information that runs counter to their narrative as “fake news” while uncritically lending credibility to spurious information. Fake news may not be 100% false, but the term applies to information reported wherein a seed of truth is abstracted heavily or combined with “alternative facts” to bend and obscure the truth to fit a narrative. This rise in the transmissibility of fake news has led to the proliferation in disinformation—misinformation deliberately spread to have financial, social or political influence. In comparison to disinformation and misinformation spread through legacy media, which was largely spread by local and national state actors within or across borders, disinformation spread through social networks may have a variety of authors, from domestic and foreign governments, extremist groups, and politicians, to business figures and local or international profiteers—those who earn money from disinformation through advertising revenue, related product sales, or capital gains from favorable movements in the prices of financial instruments. In order to limit the spread of fake news, social media sites have introduced various methods of detection, from screening out websites known for misinformation, to text detection within articles and/or metadata, to advanced methods such as text detection using optical character recognition within images. These attempts are not without their flaws, however, as over-sensitivity to specific keywords, patterns, and phrases may exist.

Previous published methods of text classification of fake news articles, titles, and opinion-related spam in general include analysis of lexical, psychological, grammatical, syntactical, and morphological features, or analysis of the relative frequency or salience of words or their stemmed forms. We propose a method of spam classification that relies not on abstracted features or content but instead a language-agnostic method that takes the individual characters as input. Though applying character-based text classification is a relatively-trivial extension of the application of convolutional neural networks to categorically-structured data, it is only now become more accessible due to recent advancements in both software development and hardware being able to handle the time and memory complexity.



News	Size (Number of articles)	Subjects	
		Type	Articles size
Real-News	21417	*World-News*	*10145*
		Politics-News	*11272*
Fake-News	23481	Type	Articles size
		Government-News	*1570*
		Middle-east	*778*
		US News	*783*
		left-news	*4459*
		politics	*6841*
		News	*9050*

Figure 1: Breakdown of topics.

Exploratory Data Analysis

The data come from the Kaggle Fake vs. Real News dataset from Ahmed, Traore, and Saad (2017) and "Ahmed et al. (2018), which contains fake and real headlines, article bodies, topics, and dates in two class-separated csv files, with 23481 fake articles, with $y = 1$ for label, and 21417 real articles with $y = 0$ for label. All of the real articles come from Reuters, while the fake articles come from a Kaggle collection assembled from articles flagged by Politifact and Wikipedia. Because the data from the fake news sites have a more diverse underlying source than the real news, it is possible that there are reader-salient elements within the genuine article bodies that may 'tip off' the classifier as to what class to which the article belongs; for example "CITY, Country (Reuters) -" is the header of most Reuters body paragraphs, and "21NEWSWIRE" is in several of the fake news paragraphs.

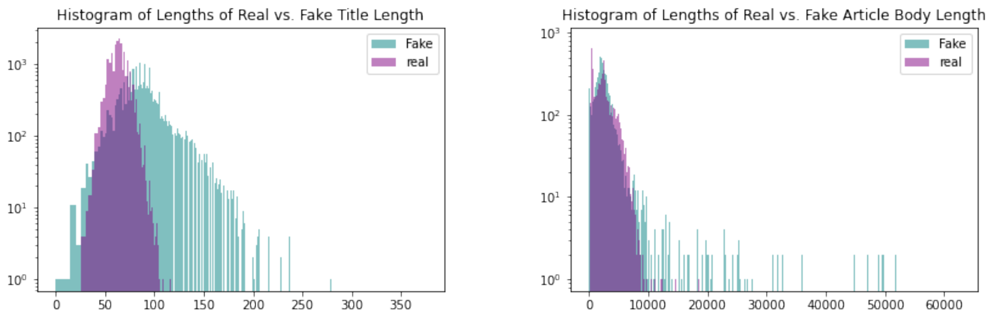


Figure 2: Histograms of real and fake titles and article texts

The real titles have more variability in length, while the distribution of fake news article body texts has a strong rightward skew. Few real article titles are shorter than 30 characters or exceed 100 characters in length, so that indicates that there may be appr/opriate boundaries within that range, so tests should be carried out with reduced data.

Literature Review

The problem of detecting fake news has a strong corollary with other types of spam detection, such as online review spam detection. Jindal and Liu (2008) divided Amazon spam comments into three classes: bogus reviews—spam intended to praise/defame the product in order to raise/lower its status within the store; reviews on brands, where the text praises/defames the brand/company behind the product with no information regarding product-specific experience; finally, non-reviews, including advertisements, questions/answers, and random texts (i.e. the comment has no sentiment, real or fake). Classification was performed using logistic regression for each task. A first-pass screening of duplicate comments can be eliminating comments with a certain level of similarity; in the case of text data, one can use the Jaccard similarity, which takes the set of all words in comments d_i, d_j and calculates $J_{i,j} = \frac{\text{unique words}(d_i) \cap \text{unique words}(d_j)}{\text{unique words}(d_i) \cup \text{unique words}(d_i)}$. In their analysis of these reviews, they found that 90% of reviewers on a subset of products had a maximum comment Jaccard similarity (vs. any reviewer) of 0.1, with the fewest having a Jaccard similarity of around 0.5, with the number slowly increasing up to a maximum score of 1, with 6% of reviewers having a maximum Jaccard score of 1. Users may either be spam accounts or genuine accounts accidentally sending duplicate reviews. In the context of fake news headlines, Jaccard similarity can be used as a distance metric in a k-nearest neighbors classifier, though this simple implementation may have bias towards classifying certain news topics or sources with certain styles (e.g. tabloids) as fake news. Brand reviews and non-reviews are the easier classification problems, as bogus reviews have a level of verisimilitude. The features used for classifying the former two types of spam include review-centric features, reviewer features, and product features, with textual features being particularly of interest. Frequency of positive and negative words in reviews, and frequency of brand name mentions, numbers, capital letters, and all-capital words were examined. Additionally, cosine similarity between reviews and product features was specifically helpful in detecting advertisements, with cosine similarity $S_{\cos}(d_i, d_j) = \frac{d_i d_j}{||d_i|| \ ||d_j||}$, with d_i being an $1 \times n, n = \#$ unique words in dataset feature vector with each entry being the count of a unique word; since large values of $S_{\cos}$ indicate high degrees of similarity in overlap of vocabulary, $1 - S_{\cos}(d_i, d_j)$ can be also used as a distance metric for classification tasks. An L_1 version using Manhattan distance or L_p using p-dimensional Minkowski distance can be formulated by replacing the L_2 distance metric with the L_p-norm. For classification of bogus reviews, the task involved looking at reviewer history to examine biases in reviews of similar products using outliers: positive reviews on poorly-reviewed products, negative reviews on well-reviewed products, and polarized reviews on products with average ratings. Jindal and Liu (2008) applied uplift modeling, a technique often found in marketing contexts, to assess likelihood of classification as spam based on four different types of reviewers: positive and negative reviewers (those $\pm 1\sigma$) in terms of sentiment, and positive and negative same-brand reviewers. Uplift per decile was implemented to compare the likelihood of positive spam classification to these four classes compared to random chance. A summary of uplift modeling and uplift per decile can be found in Gutierrez and Gérardy (2017), though the methods in this paper does not have random assignment of treatments so causality is not relevant. Reviewers who repeatedly gave negative reviews for the same brand were most likely across all deciles to be classified as spam, whereas those with positive reviews generally were less likely than random to be classified as spam.

Horne and Adali (2017) proposed a method of classification using features based on complexity, style, and intended psychological effect by using various NLP toolkits for Python, and applied an ANOVA to explore their datasets. Real news article titles are more likely on average to contain fewer proper nouns than fake or satire news article titles, while real news article bodies were more likely to contain positive words and less likely to contain negative words than fake news articles. Their Linear SVM was applied to a dataset of 75 real articles, 75 fake news articles, and 75 satire articles, with performance assessed on the mean of 5-fold cross-validation of one-versus-one classification tasks. The SVM performed best on distinguishing satire article bodies from real article bodies, and performed worst on distinguishing satire article titles from fake news article titles.

One common method of dimensionality reduction of text datasets is the term-frequency inverse-document-frequency ($tfidf(D, d, t) = tf(t, d) \times idf(t, D)$) where D is the set of all data, $d \in D$, and t are the set of terms within all documents D. The term frequency within a document can be described by the raw count divided by the total number of terms in the document $tf(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$, or through alternative measures of $tf(t,d)$ such as the raw count, binary, or through a normalization method such as log normalization $tf(t,d) = \log(1 + f_{t,d})$ or through double normalization with strength K, $tf(t,d) = K + (1-K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$. Inverse document frequency $idf(t,D)$ is a measure that increases with the rarity of a term, and is classically measured by $idf(t,D) = \log \frac{|D|}{|d \in D : t \in d|}$ where $|D|$ is the number of documents and $|d \in D : t \in d|$ is the number of documents where t can be found. Alternatives includes the smooth IDF $1 + \log \frac{|D|}{1+|d \in D : t \in d|}$ which reduces the deweighting of common document-wide terms (e.g. 'for,' 'the,' 'and') and the probabilistic IDF $\log \frac{|D| - |d \in D : t \in d|}{|d \in D : t \in d|}$ which further deweights the most common document-wide terms. TFIDF is implemented in feature extraction tasks for "Ahmed et al. (2018) and Ahmed, Traore, and Saad (2017).

Ahmed, Traore, and Saad (2017) preprocessed the data by removing unnecessary characters, tokenizing the resulting words, removing stop words—common words such as pronouns, prepositions, and infinitives, and stemming—reducing conjugated forms e.g. `'{run, runner, running, drink, drinker, drinking}` $\rightarrow$ `'{run, run, run, drink, drink, drink}`. From these, n-grams (word sequences of length n) were extracted, and the top k n-grams were selected for each sample. For the top 50,000 cleaned $n = 1$-grams (i.e. single words), the accuracy of the linear SVM was 92.0%.

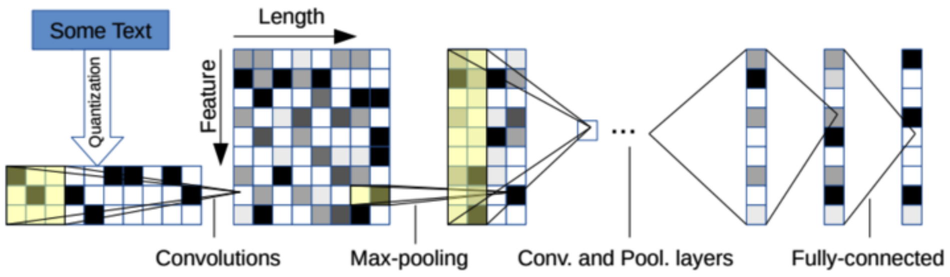
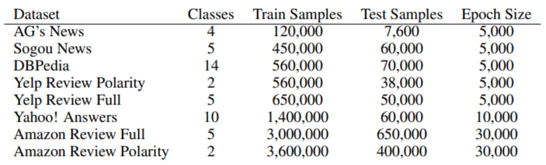


Figure 3: Character-level convolution

Zhang, Zhao, and LeCun (2016) proposed a character-level method of classification of text using 1-dimensional convolutional neural networks, allowing for flexibility with and generalization to problems with different character sets (e.g. Greek alphabet, Hangul, Emoji). The proposed method works by assigning each character (including padding and unknown characters as specific entities) as unique integer IDs which are converted to one-hot vectors within the character embedding space. Each sample is now represented by a 2D array which can be thought of as a time series of one-hot vectors. From this point, the embdedded data is convolved using several filters and max-pooled several times before flattening to a series of fully-connected layers with dropout before the final output layer, which has 1 node in binary classification and regression problems and k nodes in a k-way classification problem.

The example model structure is offered in small or large configurations in the example. Within the convolutional stage, it uses 6 rounds of ReLu-activated 1-D convolution with feature sizes of either 256 (small) or 1024 (large); the first two 1-D convolutional layers have a filter size of 7 and are both followed by a max-pooling layer with a pooling number of 3. After this second max-pooling layer, the data passes through the 3 ReLu-activated 1-D convolutional layers with filter size of 3 before the final 1-D convolutional layer, which also has a filter size of 3 and is followed by another max-pooling layer with a pooling number of 3. This feature array is flattened as it gets passed to the two dropout-regulated fully-connected layers, which are either of size 1024 (small) or 2048 (large); the dropout probability for their experiments was set to $p_{\text{dropout}} = 0.5$. In order to control for generalization error through data augmentation, on some experiments using the character-level CNN the authors implemented a weighted thesaurus method, where new samples are copied from existing samples, and the sentences are scanned for replaceable words (i.e. words with a synonym); the probability that n words are replaced in a given sample follows a geometric distribution where $P(n = x) = (1-p)p^x, x \in \mathbb{Z} \geq 0, p \in (0,1)$; when it is decided that a word is replaced, it is replaced with a synonym with index s in the similarity-sorted list which also follows a geometric distribution, with the probability of choosing index s is $P(s = y) = (1-q)q^y, y \in \mathbb{Z} \geq 0, q \in (0,1)$. This method is unfortunately agnostic to whether or not the replaced word(s) are load-bearing (i.e. their replacement by synonym(s) completely changes the meaning of a sentence). The authors set $p = q = 0.5$ for their analysis, though this provides an hyperparameter that can be further inspected. With the dataset from Ahmed, Traore, and Saad (2017) the concern of augmentation using the thesaurus method is that its random assignment of replacement words may directly affect the classification task because unusual patterns of speech are often found in spam, whereas this augmentation may be better fit for tasks such as sentiment analysis or topic/ontological classification. The character-level CNN was compared to to classical NLP methods such as bag-of-words and its TFIDF, bag-of-n-grams and its TFIDF, and bag-of-means on word embedding, as well as deep learning methods such as a long-short term memory (LSTM) network and word-based convolutional networks using word2vec or a lookup table.



Dataset	Classes	Train Samples	Test Samples	Epoch Size
AG's News	4	120,000	7,600	5,000
Sogou News	5	450,000	60,000	5,000
DBPedia	14	560,000	70,000	5,000
Yelp Review Polarity	2	560,000	38,000	5,000
Yelp Review Full	5	650,000	50,000	5,000
Yahoo! Answers	10	1,400,000	60,000	10,000
Amazon Review Full	5	3,000,000	650,000	30,000
Amazon Review Polarity	2	3,600,000	400,000	30,000

Figure 4: Comparison of classification tasks in Zhang, Zhao, and LeCun (2016). Epoch size is measured by dividing training size by batch size.

The authors' models performed the best in tasks with large sample sizes (i.e. the latter four entries in [4]), with n-gram methods performing best for the tasks smaller datasets, indicating its quality for implementation in large datasets and potential commercial applications. From their results, the authors learned that the semantics of the task do not affect the classification capability of character-level CNN and that considering uppercase characters to be separate from their lowercase counterparts was not helpful for larger datasets; in environments and learning tasks where capitalization conveys less meaning, distinguishing between the two is likely to be not as important. In the case of fake news detection, it is possible that fake news headlines or body texts use nonstandard capitalization (e.g. Phrases In Title Case, PHRASES IN ALL CAPS) and thus whether uppercase letters should be retained in the embedding ought to be tested as part of this paper's methodology.

Methodology

Computer Information

Experiments were performed on Ubuntu 20.04 running on Windows Subsystem for Linux on an 8-core, 16-thread AMD Ryzen 3700X and NVIDIA RTX 2070 SUPER, with 32GB maximum virtual memory usage. The memory limit was quickly reached with analysis of full article content, so future experimentation on a virtual machine or on upgraded hardware should be carried out to extend upon the methods in this paper.

Software

Python 3.7

Linear Algebra and Data: `NumPy`, `pandas`

Neural Network Architecture: `Tensorflow`, `keras` backend

Statistics, Classification, and Evaluation: `scikit-learn`

Other Packages: `re`, `time`

Preprocessing

For experiments where the input length was constrained, all titles of at least 30 and no more than 96 characters in length were retained, while all others were discarded. All experiments had data that were balanced by finding the greatest multiple of 100 less than or equal to the size of either class, and capping the number of samples per class to that number. From there, the quotation marks are cleaned to standard nondirectional single and double quotes, and each character is assigned an ID such that every non-whitespace character is assigned an ID, with unknown characters being assigned `UNK`; since the length of this ID table depends on the character set used, uppercase and lowercase characters are not assigned uniform IDs. These strings of IDs are padded with 0 characters to achieve a uniform length so that the model may take the information as input.

Table 1: Comparative table of models tested. L=lowercase, U=Uppercase, C=C1D=Convolution, M=MP=Max Pooling,FC=D=Fully Connected

Samples	Case	Input	Layer.Order	C1D.Filters	Kernel.Size	MP.Factor	FC.Size	Params
42800	L	339	C-M-C-M-C-C-C-C-M-D-D	256	6-6-2-2-2-2	3	1024	4786143
42800	U	339	C-M-C-M-C-C-C-C-M-D-D	256	6-6-2-2-2-2	3	1024	4837025
42800	L	339	C-M-C-M-C-C-C-C-M-D-D	256	6-6-2-2-2-2	3	256	2031327
42800	U	339	C-M-C-M-C-C-C-C-M-D-D	256	6-6-2-2-2-2	3	256	2082209
29800	L	123	C-M-C-M-C-C-C-C-M-D-D	256	6-6-2-2-2-2	3	256	1507039
29800	U	123	C-M-C-M-C-C-C-C-M-D-D	256	6-6-2-2-2-2	3	256	1557921
29800	L	123	C-M-C-M-C-C-C-C-M-D-D	256	6-6-2-2-2-2	3	64	1395871
29800	U	123	C-M-C-M-C-C-C-C-M-D-D	256	6-6-2-2-2-2	3	64	1446753

Classification

Experimentation was performed initially using a classifier with internals nearly identical to that of Zhang, Zhao, and LeCun (2016). From here, we varied the input length, character set, and number of nodes within the fully-connected layers. The activation layers are governed by the $\text{ReLU}(x) = \max\{0, x\}$ and batch size for training was set to 64 samples.

Table 2: Comparative results table of models.

Model	Train	Valid	Test
1	99.81	93.47	93.25
2	99.95	93.17	92.73
3	99.97	92.21	90.75
4	99.84	93.86	93.81
5	99.68	93.71	91.58
6	100.00	94.53	93.09
7	90.85	88.72	87.62
8	99.91	94.19	92.95

Results

Of the models including the 339-character input width and nearly-full balanced dataset, the model performing the best on the validation and testing set was the model using uppercase and lowercase letters with 256-node fully-connected layers. On the reduced 123-character input set with the reduced balanced dataset, the model using uppercase and lowercase letters with 256-node fully-connected layers also performed the best on both the validation and testing sets.

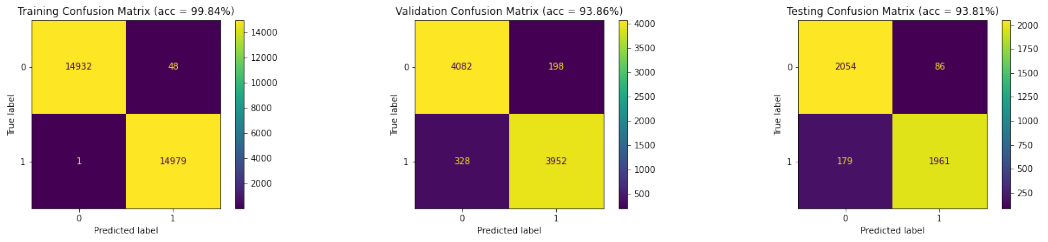


Figure 5: Confusion matrices for Model 4 (99.84%|93.86%|93.81%)

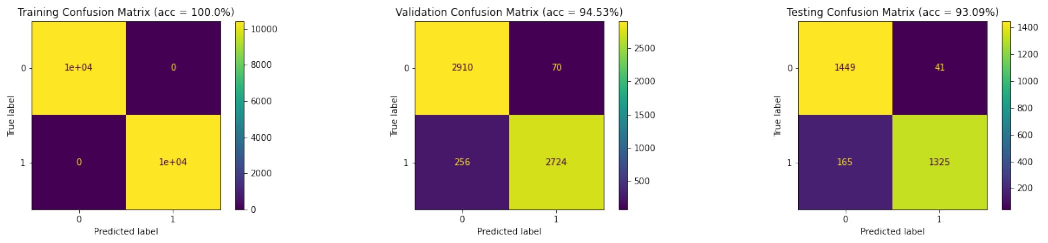


Figure 6: Confusion matrices for Model 6 (100%|94.53%|93.09%)

Conclusions and Limitations

The models could distinguish the marked differences between the real and fake titles, with consideration of capitalization potentially conferring a benefit to predictive power while incurring a relatively-small cost in computational complexity, as the number of parameters added by widening the character table has a relatively-small impact on parameter count compared to further steps in the model construction process, lending credence to its scalability and flexibility. Model structure also deserves further investigation; it is possible that a narrowed, deepened, or shortened model structure may be best, as the effects of altering other aspects of layer attributes or model structure ought to be explored. In images, the stacking of layers with smaller kernel sizes is an efficient way to detect features while cutting down on the overall number of parameters tuned. Since the convolution operation only exists in one dimension in the case of text data, we do not necessarily reap the benefit of reduced parameters with this method, but it is possible that increased classification accuracy justifies the slight increase in computational expense of a deeper network with narrower filters. Finally, considering that the real and fake data were derived from specific sources, there remains a major concern that the model may be overfitting based on detecting certain ‘signatures’ within the text that, for example, may misclassify genuine news headlines regarding a conspiracist’s-favorite topic, or may not be written in the prosaic style of Reuters; with a larger and more diverse dataset, the model can become better attuned to the underlying features within headlines, and alterations to its configuration become more meaningful.

References

"Ahmed, Hadeer, Issa Traore, editor="Traore Saad Sherif", Isaac Woungang, and Ahmed" Awad. 2018. “Detecting Opinion Spams and Fake News Using Text Classification.” *Security and Privacy* 1 (1): e9. https://doi.org/https://doi.org/10.1002/spy2.9.

Ahmed, Hadeer, Issa Traore, and Sherif Saad. 2017. “Detection of Online Fake News Using n-Gram Analysis and Machine Learning Techniques.” In *Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*, edited by Issa Traore, Isaac Woungang, and Ahmed Awad, 127–38. Cham: Springer International Publishing.

Gutierrez, Pierre, and Jean-Yves Gérardy. 2017. “Causal Inference and Uplift Modelling: A Review of the Literature.” In *Proceedings of the 3rd International Conference on Predictive Applications and APIs*, edited by Claire Hardgrove, Louis Dorard, Keiran Thompson, and Florian Douetteau, 67:1–13. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v67/gutierrez17a.html.

Horne, Benjamin D., and Sibel Adali. 2017. “This Just in: Fake News Packs a Lot in Title, Uses Simpler, Repetitive Content in Text Body, More Similar to Satire Than Real News.” https://arxiv.org/abs/1703.09398.

Jindal, Nitin, and Bing Liu. 2008. “Opinion Spam and Analysis.” In. https://doi.org/10.1145/1341531.1341560.

Zhang, Xiang, Junbo Zhao, and Yann LeCun. 2016. “Character-Level Convolutional Networks for Text Classification.” https://arxiv.org/abs/1509.01626.

Appendix A: Code

Required Packages

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, Embedding, Activation, Flatten, Dense
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Dropout
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.utils import plot_model
import math
import time
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import re
import os
import os
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.python.client import device_lib
# Model saving
import bz2
import pickle
import _pickle as cPickle
# Saves the "data" with the "title" and adds the .pickle
def full_pickle(title, data):
    pikd = open(title + '.pickle', 'wb')
    pickle.dump(data, pikd)
    pikd.close()
def loosen(file):
    pikd = open(file, 'rb')
    data = pickle.load(pikd)
    pikd.close()
    return data
# Pickle a file and then compress it into a file with extension
def compressed_pickle(title, data):
    with bz2.BZ2File(title + '.pbz2', 'w') as f:
        cPickle.dump(data, f)
def decompress_pickle(file):
    data = bz2.BZ2File(file, 'rb')
    data = cPickle.load(data)
    return data
device_lib.list_local_devices()
```

Fake News Code

```
class fakeNews:
    def __init__(self,directory='/mnt/c/Users/thecu/Documents/R Projects/4270/charLevelCNN/',lowercase=
        self.lowercase = lowercase
        self.directory = directory
        if lowercase == True:
            self.alphabet =
            "abcdefghijklmnopqrstuvwxyz0123456789,;.!?:'\"/\\|_@#$%^&*~`+-=<>()[]{}"
        else:
            self.alphabet =
            "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789,;.!?:'\"/\\|_@#$%^&*~`+-=<>
    def make_fakeNews(self, balance=True, content = False, pad = 1.25):
        self.pad = pad
        self.balance = balance
        self.content = content
        self.pad = pad
        articles = []
        for dirname, _, filenames in os.walk(self.directory+'data'):
            for filename in filenames:
                mypath = os.path.join(dirname, filename)
                name = re.sub('\.csv','',filename + '_all')

            exec("%s = pd.read_csv(" % (name,)+
            "'{path}".format(path=mypath)+"')")
            if name == 'Fake_all':
                booler = 1
            else:
                booler = 0
            exec("%s.insert(%s.shape[1],'y',[%d]*%s.shape[0])" % (name,name,booler,name))
            exec("articles.append(%s)" % (name,))
    if self.balance == True:
        clsLen = []
        for df in articles:
            clsLen.append(df.shape[0])
        classSize = int(np.min(100*np.floor(np.array(clsLen)/100)))
        articles_balanced = []
        for df in articles:
            articles_balanced.append(df.iloc[:classSize,:])
        Articles_bal = pd.concat(articles_balanced,axis=0,ignore_index=True)
        Articles_all = Articles_bal
    else:
        Articles_ubl = pd.concat(articles,axis=0,ignore_index=True)
        Articles_all = Articles_ubl
    tk = Tokenizer(num_words=None, char_level=True, oov_token='UNK')
    train_titles = Articles_all.iloc[:,0]
    train_bodies = Articles_all.iloc[:,1]
    all_y = Articles_all['y'].values
    tk.fit_on_texts(train_titles)
    char_dict = {}
    for i, char in enumerate(self.alphabet):
        char_dict[char] = i + 1

    # Use char_dict to replace the tk.word_index
```

```
        tk.word_index = char_dict.copy()
        # Add 'UNK' to the vocabulary
        tk.word_index[tk.oov_token] = max(char_dict.values()) + 1
        # Convert string to index
        title_sequences = tk.texts_to_sequences(train_titles)
        max_ttl = int(math.ceil(self.pad*max([len(x) for x in Articles_all.iloc[:,0].values])))
        if self.content == True:
            content_sequences = tk.texts_to_sequences(train_bodies)
            max_cnt = int(math.ceil(self.pad*max([len(x) for x in Articles_all.iloc[:,1].values])))
            coded_titles = pad_sequences(title_sequences, maxlen=max_ttl,
            padding='post')
            coded_bodies = pad_sequences(content_sequences, maxlen=max_cnt,
            padding='post')
            return (np.concatenate([coded_titles,coded_bodies],axis=1).astype('int32'),
            all_y,tk.word_index)
        else:
            return (pad_sequences(title_sequences, maxlen=max_ttl, padding='post').astype('int32'), all_y,t
    def three_way_split(self,make_Fake, split='default'):
        self.X = make_Fake[0]
        self.y = make_Fake[1]
        tot_len = self.y.shape[0]
        self.split = split
        if self.split == 'default':
            tts = (7,2,1)
        else:
            tts = self.split
        tsp = int((tts[2]*tot_len/20))
        vsp = int(((tts[1]+tts[2])*tot_len/20))
        hlf = int((1*tot_len/2))
        test_tp =  (np.concatenate([self.X[:tsp,:],self.X[hlf:(hlf+tsp),:]],axis=0),
        np.concatenate([self.y[:tsp],self.y[hlf:(hlf+tsp)]],axis=0))
        valid_tp = (np.concatenate([self.X[tsp:vsp,:],self.X[(hlf+tsp):(hlf+vsp),:]],axis=0),
        np.concatenate([self.y[tsp:vsp], self.y[(hlf+tsp):(hlf+vsp)]],axis=0))
        train_tp = (np.concatenate([self.X[vsp:hlf,:],self.X[(hlf+vsp):,:]],axis=0),
        np.concatenate([self.y[vsp:hlf],self.y[(hlf+vsp):]],axis=0))
        app = []
        for tp in [train_tp, valid_tp,test_tp]:
            alength = np.arange(tp[1].shape[0])
            np.random.seed(5318008)
            np.random.shuffle(alength)
            wee = []
            for q in tp:
                if len(q.shape) == 2:
                    wee.append(q[alength,:])
                elif len(q.shape) == 1:
                    wee.append(q[alength])
            app.append(wee)
        return app
    def makeClassifier(self,make_Fake, fully_connected_layers = None, conv_layers = None, dropout = 0.5, op
        self.fully_connected_layers = fully_connected_layers
        self.dropout_p = dropout
        self.optimizer = optimizer
        self.loss = loss
        self.X = make_Fake[0]
```

```
self.y = make_Fake[1]
self.wind = make_Fake[2]
self.fully_connected_layers = fully_connected_layers
self.conv_layers = conv_layers
input_size = self.X.shape[1]
vocab_size = len(self.alphabet) + 1
embedding_size = len(self.alphabet) + 1
if self.fully_connected_layers == None:
    fully_connected_layers=[1024, 1024]
if self.conv_layers == None:
    conv_layers = [[256, 7, 3],
           [256, 7, 3],
           [256, 3, -1],
           [256, 3, -1],
           [256, 3, -1],
           [256, 3, 3]]
        # Embedding weights
embedding_weights = []  # (len(alphabet)+2, len(alphabet)+1)
embedding_weights.append(np.zeros(vocab_size))  # (0, len(alphabet)+2)

for char, i in self.wind.items():  # from index 1 to len(alphabet)+2
    onehot = np.zeros(vocab_size)
    onehot[i - 1] = 1
    embedding_weights.append(onehot)

embedding_weights = np.array(embedding_weights)
print('Load')

# Embedding layer Initialization
embedding_layer = Embedding(vocab_size + 1,
                            embedding_size,
                            input_length=input_size,
                            weights=[embedding_weights])

# Model Construction
# Input
inputs = Input(shape=(input_size,), name='input', dtype='int64')         # shape=(?, padded input si
# Embedding
x = embedding_layer(inputs)
# Conv
for filter_num, filter_size, pooling_size in conv_layers:
    x = Conv1D(filter_num, filter_size)(x)
    x = Activation('relu')(x)
    if pooling_size != -1:
        x = MaxPooling1D(pool_size=pooling_size)(x)  # Final shape=(None, 34, 256)
x = Flatten()(x)  # (None, 8704)
# Fully connected layers
for dense_size in fully_connected_layers:
    x = Dense(dense_size, activation='relu')(x)  # dense_size == 1024
    x = Dropout(self.dropout_p)(x)
# Output Layer
predictions = Dense(1, activation='sigmoid')(x)
# Build model
model = Model(inputs=inputs, outputs=predictions)
```

```
    model.compile(optimizer=optimizer, loss=loss,
    metrics=['accuracy'])  # Adam, categorical_crossentropy
    model.summary()
    return model
    #plot_model(model, to_file=self.directory + 'model_plot.png', show_shapes=True, show_layer_names=Tru
```

Histograms

```
scrimblo = fakeNews(lowercase=True)
edastart = scrimblo.make_fakeNews(eda=True)
bseq = np.sqrt(np.arange(626))
for q in range(len(edastart)):
    if edastart[q]['y'].values[0]==1:
        lbl = 'Fake'
        col = '#007f7f7f'
    else:
        lbl = 'real'
        col = '#7f007f7f'
    plt.hist(np.ravel(edastart[q]['title'].apply(lambda x: len(x)).values),bins=15*bseq,color=col, label
plt.legend()
plt.title('Histogram of Lengths of Real vs. Fake Title Length')
plt.savefig('/mnt/c/Users/thecu/Documents/R Projects/4270/charLevelCNN/eda/' + 'titles_hist.png')
plt.clf()
for q in range(len(edastart)):
    if edastart[q]['y'].values[0]==1:
        lbl = 'Fake'
        col = '#007f7f7f'
    else:
        lbl = 'real'
        col = '#7f007f7f'
    plt.hist(np.ravel(edastart[q]['text'].apply(lambda x: len(x)).values),bins=2500*(25-bseq)[::-1],colo
plt.legend()
plt.title('Histogram of Lengths of Real vs. Fake Article Body Length')
plt.savefig(
'/mnt/c/Users/thecu/Documents/R Projects/4270/charLevelCNN/eda/' + 'bodies_hist.png')
```

Models

Model 4

```
    scrimblo_u = fakeNews(lowercase=False)
zabloing_uf = scrimblo_u.make_fakeNews(content=False,raw=False)
scrungus_ud = scrimblo_u.makeClassifier(zabloing_uf,fully_connected_layers=[256,256])
kablooey_uf = scrimblo_u.three_way_split(zabloing_uf)
myModel3 = scrimblo_u.get_fit(scrungus_ud,kablooey_uf, name='model_3',
          batch_size = 64,
          epochs = 40,
          verbose = 2,
          display = True,
          predict = True)
```

Model 6

```
scrimblo_u = fakeNews(lowercase=False)
zabloing_uf = scrimblo_u.make_fakeNews(content=False,raw=False,min_title_len=30,max_title_len=96,x_wid=
scrungus_ud = scrimblo_u.makeClassifier(zabloing_uf,fully_connected_layers=[256,256])
kablooey_uf = scrimblo_u.three_way_split(zabloing_uf)
myModel5 = scrimblo_u.get_fit(scrungus_ud,kablooey_uf, name='model_5',
          batch_size = 64,
          epochs = 40,
          verbose = 2,
          display = True,
          predict = True)
```