THE CATHOLIC UNIVERSITY OF AMERICA

Final Project – Detecting Fake News

Rafael A. Moreno Contreras
Applications of Data Analytics and Development DA516
The Catholic University of America

Introduction

Do we trust all the news we hear from social media? All news is not real, so how can we detect the fake ones? The answer is **Natural Language Processing (NLP)**.

Data generated from articles, declarations or even tweets are examples of unstructured data. **Unstructured data** doesn't fit neatly into the traditional row and column structure of relational databases, and represent the vast majority of data available in the actual world. It is messy and hard to manipulate. Nevertheless, thanks to the advances in disciplines like machine learning a big revolution is going on regarding this topic.

Nowadays it is no longer about trying to interpret a text or speech based on its keywords (the old fashioned, mechanical way), but about understanding the meaning behind those words (the cognitive way). This way it is possible to detect figures of speech like irony, or even perform sentiment analysis.

NLP is a discipline that focuses on the **interaction between data science and human language**, and is scaling to lots of industries. Today NLP is booming thanks to the huge improvements in the access to data and the increase in computational power, which are allowing practitioners to achieve meaningful results in areas like healthcare, media, finance and human resources, among others.

What is Fake News?

A type of yellow journalism, fake news encapsulates pieces of news that may be hoaxes and is generally spread through social media and other online media. This is often done to further or impose certain ideas and is often achieved with political agendas. Such news items may contain false and/or exaggerated claims, and may end up being viralized by algorithms, and users may end up in a filter bubble.

The Dataset

This dataset has a shape of 7796×4. The first column identifies the news, the second and third are the title and text, and the fourth column has labels denoting whether the news is REAL or FAKE.

		title	text	label
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL

TfidfVectorizer

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Machine learning algorithms cannot work with raw text directly. Rather, the text must be converted into vectors of numbers. In natural language processing, a common technique for extracting features from text is to place all of the words that occur in the text in a bucket. This approach is called a **bag of words** model. It's referred to as a "bag" of words because any information about the structure of the sentence is lost.

The problem with the bag of words approach is that **it doesn't account for noise**. In other words, certain words are used to formulate sentences but do not add any semantic meaning to the text. On the other hand, words like "good" and "awesome" could be used to determine whether a rating was positive or not.

Often times, when building a model with the goal of understanding text, you'll see all of stop words being removed. Another strategy is to score the relative importance of words using **TF-IDF**.

TF (Term Frequency):

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Term Frequency is the number of times a word appears in a document. A higher value means a term appears more often than others, and so, the document is a good match when the term is part of the search terms.

IDF (Inverse Document Frequency):

$$idf(w) = log(\frac{N}{df_t})$$

Words that occur many times in a document, but also occur many times in many others, may be irrelevant. IDF is a measure of how significant a term is in the entire corpus.

The TfidfVectorizer converts a collection of raw documents into a matrix of TF-IDF features.

What is a PassiveAggressiveClassifier?

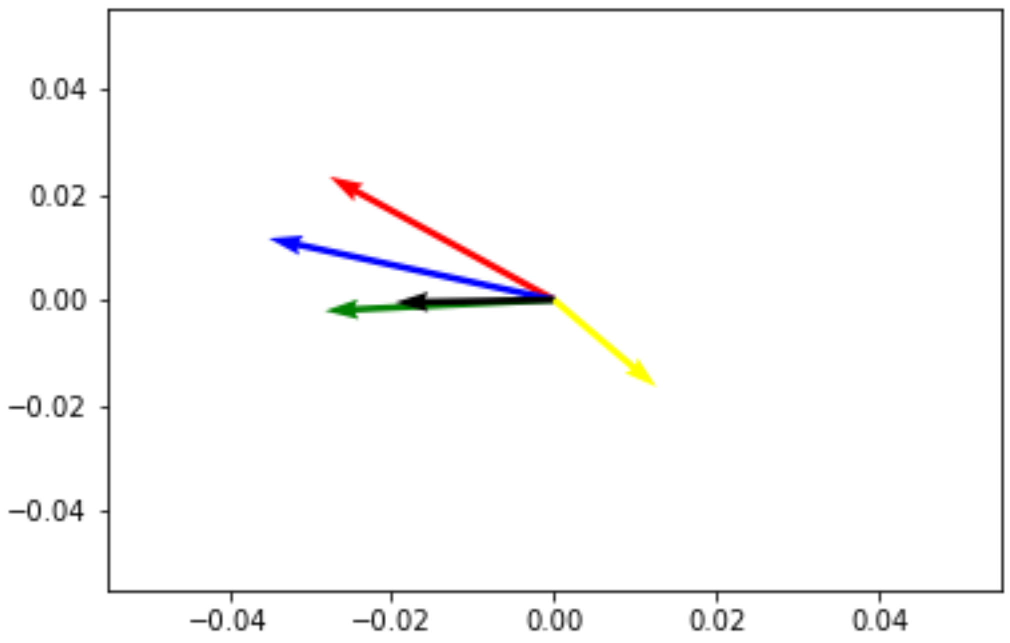
Passive Aggressive algorithms are online learning algorithms. Such an algorithm remains passive for a correct classification outcome, and turns aggressive in the event of a miscalculation, updating and adjusting. Unlike most other algorithms, it does not converge. Its purpose is to make updates that correct the loss, causing very little change in the norm of the weight vector.

After applying the PassiveAggressiveClassifier we obtained an accuracy 92.98%. With a confusion matrix as follows:

591	47
42	587

Convolutional Neural Network

Deep neural networks take a very different approach to document classification. Firstly, words are represented as embedding vectors with the idea that two words that are semantically similar to each other have similar vectors. Consider the following figure:



This figure represents a 2D embedding space with five embedding vectors each representing a different word:

- Red — Queen
- Blue — King
- Green — Man
- Black — Woman
- Yellow — Oil

We can see that concepts that are similar to each other are close together (e.g. man and woman) in this embedding space and concepts not related are further apart (e.g. oil).

Secondly, in deep neural networks documents are no longer compressed into a vector representation of just word occurrences. Instead, deep neural networks process actual sequences of words (coded as integers) as they appear in the documents thereby maintaining the order and contexts of words.

Consider the following code snippet using the Keras tokenizer applied to two sentences:

```
from keras.preprocessing.text import Tokenizer
tok = Tokenizer()
# train tokenizer
tok.fit_on_texts(["it was not good, it was actually quite bad"])
# print sequences
print(tok.texts_to_sequences(["it was not good, it was actually
quite bad"])[0])
print(tok.texts_to_sequences(["it was not bad, it was actually quite
good"])[0])
```

This will print out the following sequences:

```
[1, 2, 3, 4, 1, 2, 5, 6, 7]
[1, 2, 3, 7, 1, 2, 5, 6, 4]
```

with a word_index of:

```
{'it': 1, 'was': 2, 'not': 3, 'good': 4, 'actually': 5, 'quite': 6,
'bad': 7}
```

These sequences can be directly fed into a deep neural network for training and classification. Notice that word order and context are nicely preserved in this representation.

Our **CNN** can be broken down into two distinct parts. The first part consists of three layers and is responsible for word and sequence processing:

- The Embedding layer — learn word embeddings.
- The Convolution layer — learn patterns throughout the text sequences.
- The Pooling layer — filter out the interesting sequence patterns.

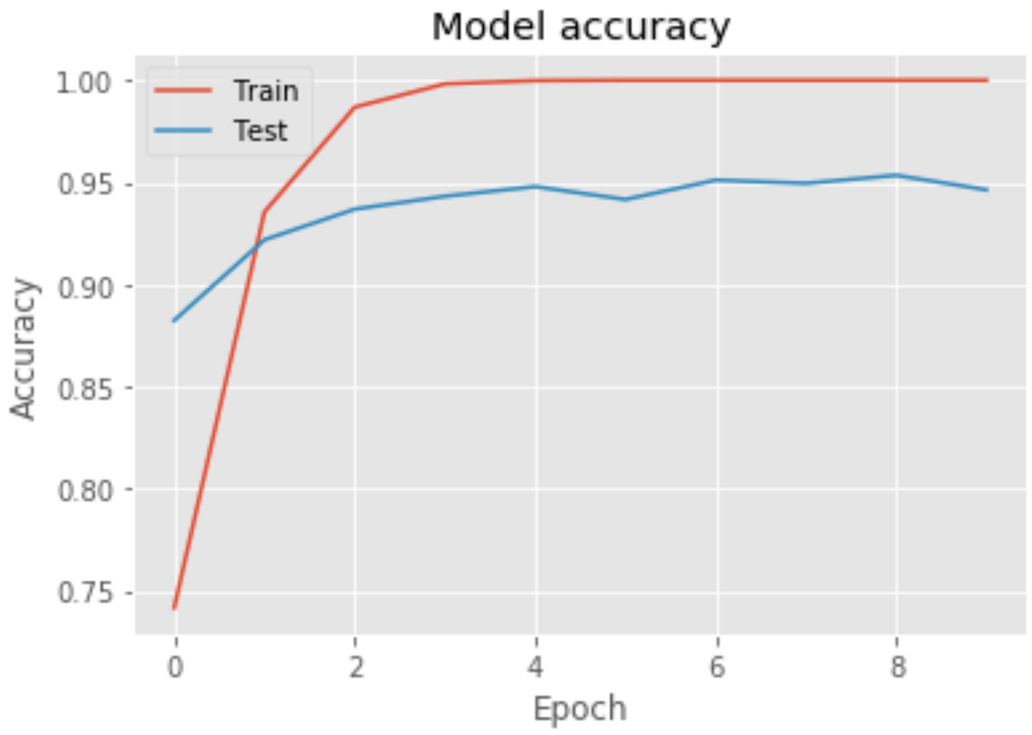
The second part consists of two layers:

- A Dense layer with a ReLU activation function.
- A Dense layer (also the output layer) with a Sigmoid activation function.

This part of the CNN can be viewed as a traditional feed-foward, back-propagation neural network with one hidden layer operating on a feature vector of length 128 computed by the first part of the CNN.

After applying the Convolutional Neural Network we obtain the following results:

```
Epoch 10/10
5068/5068 [==============================] - 735s
145ms/step - loss: 9.2196e-06 - acc: 1.0000 - val_loss:
0.2261 - val_acc: 0.9463
```



We can see there is some overfitting on the model, this could possible be corrected by further twitching the parameters, but given the amount of words the model has to tokenize, compare and analyze, each epoch ends up taking between 10 and 15 min. This makes experimentation very time consuming, but without time constraints the performance could be improved.

Summary

We learned how to detect fake news, we took a political dataset, implemented a TfidfVectorizer, initialized a PassiveAggressiveClassifier, and fit our model. We ended up obtaining an accuracy of 92.82%.

After we trained our network for 10 epochs with a batch size of 128 using an 80–20 training/hold-out set. A couple of notes on additional parameters:

- The vast majority of documents in this collection is of length 5000 or less. So for the maximum input sequence length for the DNN I chose 5000 words.
- There are roughly 100,000 unique words in this collection of documents. I arbitrarily limited the dictionary that the DNN can learn to 25% of that: 25,000 words.
- Finally, for the embedding dimension, I chose 300 simply because that is the default embedding dimension for both word2vec and GloVe.

The results were very good, around 95% accuracy for the CNN model.

The performance increase is statistically significant compared to the performance of the PasiveAggressiveClassifier, but it comes with a cost, each epoch takes between 10 to 15 min, so testing iterations can take a whole day.

One conclusion that one might draw is that semantic similarity between words and word order or context are crucial for document classification.