GEORGE TRASKAS

Fake News Detector Proposal

Prepared for: Udacity ML Engineer Nanodegree
Prepared by: George Traskas, Data Scientist
15 June 2020

EXECUTIVE SUMMARY

Objective

The term 'fake news' regularly hits the headlines nowadays. Whether it is to do with political events or information on the various social platforms, it seems like it is getting harder and harder to know who to trust to be a reliable source of information. This project is about understanding of what is actually fake news and propose an efficient way to detect them using the latest machine learning and natural language techniques.

Goals

The project's desired results are:

- a model in which text can be provided as input and
- predict if it is fake or true.

Datasets

A dataset from Kaggle was used as a training, validation, and test input for the algorithms that were used. The dataset features a list of articles, together with the subject of the article and its title categorised as 'Fake' or 'True'.

Acknowledgements for the dataset:

- Ahmed H, Traore I, Saad S. "Detecting opinion spams and fake news using text classification", Journal of Security and Privacy, Volume 1, Issue 1, Wiley, January/February 2018.

- Ahmed H, Traore I, Saad S. (2017) "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques. In: Traore I., Woungang I., Awad A. (eds) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017. Lecture Notes in Computer Science, vol 10618. Springer, Cham (pp. 127-138).

Proposed Solution

It is more wise to first create a Proof of Concept (PoC) locally in our machine (even with a fraction of data) in order to demonstrate that the proposed solution has a practical potential. Then, we can proceed to an end to end solution deploying a ML model in AWS.

For the implementation:

- Explore data to derive useful insights and get a better feeling of the data (**EDA**).
- Clean, simplify, and **prepare the dataset** in a proper format:
  - Convert the entire document's text into **lower case**, so that capitalisation is ignored (e.g., IndIcaTE is treated the same as Indicate).
  - **Normalise numbers**, i.e. replace all numbers with the text "number".
  - **Remove non-words and punctuation**, as well as trim all white spaces (tabs, newlines, spaces) to a single space character.
  - **Tokenise**, i.e. break up sequence of strings into pieces, such as words, keywords, phrases, symbols and other elements called tokens. Tokens can be individual words, phrases or even whole sentences. In the process of tokenisation, some characters like punctuation marks are discarded.
  - **Stem words**, i.e. reduce words to their stemmed form. For instance, "discount", "discounts", "discounted" and "discounting" will be all replaced with "discount".
  - Apply other normalising techniques and test models for the best combinations.
- Implement a **bag-of-words** model.

  Since we cannot work with text directly when using machine learning algorithms, we need to convert the text to numbers. Algorithms **take vectors of numbers as input**, therefore we need to convert documents to fixed-length vectors of numbers. A simple and effective model for thinking about text documents in machine learning is called the bag-of-words model, or BoW.

  The model is simple in that it throws away all of the order information in the words and focuses on the occurrence of words in a document. This can be done by **assigning each word a unique number**. Then, any document we see can be encoded as a fixed-length vector with the length of the vocabulary of known words. The value in each position in the vector could be filled with a count or frequency of each word in the encoded document.

  This is the bag of words model, where we are only concerned with encoding schemes that represent what words are present or the degree to which they are present in encoded documents without any information about order.

  The Scikit-learn library provides 3 different schemes that we could use: **CountVectorizer, TfidfVectorizer, HashingVectorizer**.
- **Split** the dataset to train, validation, and test sets.
- **Train** a fast Naive Bayes model offered by Sklearn module to get some first results from our data. This can be done locally as a PoC before proceeding to a paid service for deployment, such as AWS.
- **Evaluate** performance.
- Prepare data for **AWS SageMaker BlazingText** algorithm.
- Train the model and save its artefacts in **S3**.

- Deploy the model in another instance.
- Test with the unseen test data located in S3. Evaluate with accuracy.
- Create a **Lambda** function that prepares data/text input from a **REST API**.
- Test everything deploying a simple **Web App** html. The endpoint will be called from a simple HTML webpage. In this page, the user will be able to post a text and check whether is true or fake news.

BUDGET

Cost of Phases

For better controlling the project's advancement, I propose the following milestones with estimated required working hours.

Description	Hours	Hourly Rate	Cost
Retrieve and prepare data	10	US$ 25	US$ 250
Count word frequency and create bag-of-words model	5	US$ 25	US$ 125
Train, test, and evaluate algorithm locally	5	US$ 25	US$ 125
Start a SageMaker service and prepare data	4	US$ 25	US$ 100
Train BlazingTest model	2	US$ 25	US$ 50
Test model - caution to prepare data correctly in batches	5	US$ 25	US$ 125
Create an AWS Lambda function	5	US$ 25	US$ 125
Setup the AWS API Gateway	2	US$ 25	US$ 50
Test Web App with real data	2	US$ 25	US$ 50
Total (at max)			**US$ 1,000**

Sincerely,

George Traskas
Thessaloniki, Greece
georgiost77@gmail.com