Exact Fact: A Chrome Extension that Detects Misinformation

Daniel Allex, Abhay Sheshadri, and Kevin Liu
December 2020

Introduction

Misinformation has always been a threat to the well-being of humanity; without knowledge of the state of affairs in the outside world, how can we decide what actions to take to improve our lives? In the age of the internet, misinformation has found new methods to spread at unprecedented rates. Social media and blog sites greatly lower the bar necessary to publish information for the world to see. While the democratization of the control of information has done great good, it has also done great damage as malicious agents also have access to these resources. In order to mitigate the spread of misinformation, it is necessary to use technology to aid in the process of detecting and changing false information. To that end, we have decided to make a chrome extension that is both easy to use for every internet user and is capable of highlighting misinformation on websites.

Competition

In recent years, we have seen social media companies take an active role in combating these damaging lies. For example, youtube embeds wikipedia articles into their websites when searching for controversial topics such as climate change. Twitter has also taken a more active role by tagging posts containing misinformation.

What differentiates our approach from these services is that we want our program to apply universally to all websites, not just a select few. Although this task is considerably harder, we believe that we can do a decent job. We would also like there to be a better balance between giving broad overviews to these topics (YouTube's system) and having manually tagged posts with hyper-specific information (Twitter).

Two systems, similar to our project, have been proposed by fullfact.org and by students at the University of Texas at Arlington. Both systems rely on the use of artificial intelligence to streamline the collection of statements to be fact-checked to make the lives of professional fact-checkers easier.

Our Sources

We compare claims to a database of false claims from Politifact and Snopes. Politifact is a fact-checking website that has its primary focus on political claims. Snopes focuses more on general claims, giving us a good balance of general and political claims in the database. In addition to these sources, we have a website where users can submit false claims along with the reason for it being wrong. Other users can vote on these submissions and highly voted claims can be reviewed and added by admins to ensure accuracy.

We also detect journalists with political leanings when they are mentioned, using a list of journalists by AllSides. AllSides is a website that researches journalists and websites to determine their leanings. To ensure that these researched assessments are valid, users can vote on whether they agree with the ratings on the AllSides website, which we also display. Finally, we use Media Bias Fact Check to determine the leanings of specific websites, listed on the site.

Backend

Objective

At the core of our project, we have a RESTful API that is capable of returning misinformative sentences, as well as sources, given some text. This text would usually be the inner-text of a website. The API would accept requests from the extension and return JSON objects with the relevant information.

Methods

Sentence Segmentation

We consider a sentence to be a fundamental component of a text we are evaluating, as a sentence can contain a claim that is worth fact checking. The API splits the text into sentences using SpaCy, a natural language toolkit. SpaCy pipeline allows for quick sentence boundary detection (it is the current state-of-the-art in terms of speed) that is both accurate and robust. It is able to parse text faster than 99% of other packages for its given level of accuracy [1]. In addition, SpaCy also allows the use of other packages which provide additional utilities such as huggingface's coreference resolution plugin [2]. As sentences may use pronouns to refer to aforementioned subjects in the text, replacing corefering references to a subject with the most relevant mention is important to properly fact check certain sentences. In addition to segmenting the text into sentences, we also extract sentence fragments. Sentence fragments are clauses, including a subject-verb pair, along with some additional information. In order to do this, we use an implementation of the ClausIE algorithm [3] that relies on SpaCy's dependency parser [4]. If a sentence contains sentence fragments, we include the sentence fragments as separate sentences to be fact-checked individually.

Sentence Encoding

In order to efficiently store sentences and compare them to one another, we use a neural network to map each sentence to a vector within a high-dimensional vector space. We use a bidirectional transformer network for this task as they have achieved state-of-the-art results in text similarity [5]. Specifically, we use

DistillBert, a version of the popular transformer BERT that is more condensed and has fewer parameters, as it runs faster and is easier to train.

The Database

In order to allow for the quick retrieval of relevant sentences in a relational database given an embedding vector of a base sentence, we used an algorithm similar to what Spotify uses for their song recommendations. We create a database of false claims collected by major news sources such as PolitiFact and Snopes. The database stores claim data, as well as an embedding vector that represents that data. We then create binary trees by repeatedly partitioning the database by checking what side of a random hyperplane a vector is on. We create several trees to improve the accuracy of this approximate nearest neighbor task. This allows to query the database with incredible speed and reasonable accuracy with little memory usage.

Extension

Extension Structure

The popup from the context action service starts up and sends requests to the background script. The background script is created once the extension loads, and the same background script is used until the extension is removed. The background script manages cooldowns so that the user has to wait 10 seconds before sending a new request, even across separate web pages. The popup script and display resets every time the pinned icon is clicked. As a result, the extension sends initial requests upon starting each time. For instance, a request is sent to the background script to get cooldown data, a request is sent to the content script on a given page to get data on already collected misinformation, and a javascript file containing website leanings is consulted to compare the website host with the data. This means that before sending a request, the website leaning, any previously started cooldown, and any already found misinformation percent will be displayed. After, when clicking the scan page button, a request is first sent to the server. Then, while waiting for the server to send back data, a list of journalist names are checked to see if they are on the page. This list is saved so that after the server gives the data and misinformation is highlighted, journalist bias leanings can be highlighted as well. Finally, boxes are created but hidden to then be positioned and visible upon hovering over a highlight.

Tooltips

Adding tooltips to each marked element allows the extension to provide the user with an explanation as to what the misinformation or political bias is, sources to back up the truth, and acknowledgements that different information might have been the truth at the

time of publication of the original text or article. HTML provides an in-house solution to creating tooltips: the title attribute; however, creating a custom tooltip using the onmouseover and onmouseout events in JavaScript allowed for a simpler implementation that was more aesthetically customizable using CSS. Thus, we created a separate class that dictated the CSS of all tooltips. This still left us with the freedom to diversify how our tooltips would look in the future as tooltips regarding one category of content can be given an additional class on top of the base CSS class we gave to all tooltips. Mark.JS provides options that apply to the element marked through its mark() method. By utilizing the "each" option, every separate string that was marked would also be given a custom tooltip with text that is dependent on the marked string. Each tooltip is constructed using a div element that is appended to the marked element. Contained within the tooltip is a header and the list of sources that support the truth. The marked elements themselves are connected to the onmouseover and onmouseout events since the tooltip provides information regarding the text in the element, and a user would not want to see information about text they aren't paying attention to. Whenever the user moves their cursor over a marked element, its corresponding tooltip changes its display style from none to block to make it visible. The opposite occurs whenever the user moves their cursor off a marked element: its tooltip changes its display style from display to none to hide it. Tooltips are positioned by utilizing the coordinate data provided by the window's X and Y page offset as well as the getBoundingClientRect() method which provides the X and Y coordinates of an element relative to the viewport. The resulting coordinates are dynamic and position each tooltip below their corresponding element regardless of where the viewport currently is and what it displays. The extension also keeps track of whether a tooltip of a certain type is currently being shown. This could be either "Misinformation" or "Journalist" as they have distinct colors associated with them. If a tooltip of one type is being shown, the other type will be blocked from being shown and vise-versa. For marked text that is within another mark, this allows users to individually view the stacked tooltips as opposed to one of them visually blocking the other.

Website

Website Functionality

The website allows users to manually contribute information to the service. This would automatically increase the accuracy of the service as it has a larger set of misinformation to check future claims against. In the requests to add misinformation to the database, users provide all the necessary fields to properly organize the misinformation within the database. The website has a forum-like interface where users make posts with their requests. Other users can comment on these posts including a tag of whether they agree with the request, along with some additional information to back up their choice. We

have also implemented a button that can automatically add a post directly into our database if an user with an admin account clicks it.

Website Implementation

The website was implemented on a Flask webapp. We use jinja templating to make our website serve users dynamic content. Account information is stored within a sqlite database. Passwords are hashed using the Bcrypt algorithm before they are stored.

Product Future

Knowledge Graph

Using more advanced natural language processing methods, we intend to develop a graph of subjects and propositions relevant to them. This graph would be generated as we extract information from the internet. We would them develop an algorithm to find the subgraph relevant to a search query. The search query will be in the form of a claim that we would like to factcheck. We will then use a Graph Transformer network for a natural language inference task between the subgraph and unverified claim. Nodes in this graph would be subjects of propositions. There would be various connections between these subject nodes indicating their relationships to one another. Each subject node will also have several properties representing the context that they can be used in. We would need to restructure our database to more efficiently pull information from such a graph. A potential alternative to our knowledge graph would be an anti-knowledge graph that only has the answer to false propositions. This would allow the format to be similar to our current one.

Citations

[1] Facts & Figures · spaCy Usage Documentation. (n.d.). Retrieved September 30, 2020, from https://spacy.io/usage/facts-figures

[2] Wolf, T. (2017, July 7). State-of-the-art neural coreference resolution for chatbots [Web log post]. Retrieved September 30, 2020, from https://medium.com/huggingface/state-of-the-art-neural-coreference-resolution-for-chatbots-3302365dcf30

[3] Del Corro Luciano, and Rainer Gemulla: "Clausie: clause-based open information extraction." Proceedings of the 22nd international conference on World Wide Web. ACM, 2013.

[4] Chourdakis, E., & Reiss, J. (2018). Grammar Informed Sound Effect Retrieval for Soundscape Generation. In DMRN+ 13: Digital Music Research Network One-day Workshop (pp. 9)

[5] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.