Fake News?

Fact-Checking Individual Statements With Web Scraping and Sentiment Analysis

Sofia Escoto

August 16, 2022

1 Introduction

The spread of misinformation is a large societal problem, exacerbated by the widespread use of social media. Today, anyone can log into their computer and say something that the entire world can see. While the malicious spread of misinformation is often seen in regards to politics, it's not exclusive to it, and though anyone with a computer can also access a search engine and tell if a statement is false based on the results, it's rare that a person will take the time out of their day to do so. Because of this, many people spread false, and potentially harmful, statements online due to ignorance.

With a machine learning algorithm that takes a statement (ex. a blue whale can weigh up to 200 tons) and classifies the statement as either true or false, a person doesn't have to take any time to Google it on their own. With this model used in conjunction with or integrated into websites like Twitter, Facebook, and Reddit, misinformation could be easily recognized as false and therefore would be unlikely to "go viral" and spread to more users.

Though machine learning has been used before for fact-checking and determining the credibility of a claim, most of its application has been in analyzing either full-length articles using context clues and sources or on subjective statements that were trained to use 'common sense'. Obtaining truth values of objective statements has not been previously attempted in the public sphere, (i.e. outside of scientific claims [1]) or without significant amounts of pre-validated data available (i.e. as a part of a broader algorithm with a question-and-answer input-output model that relies on fact databases [2]).

With this proposed model, the truth values of objective statements will be available with a single click to ordinary Internet users, who may have otherwise just taken whatever they read online at face value.

2 Literature Review

Most often when a person wants to know whether something online is true or not, they're concerned with political articles. Using machine learning to analyze fake news (which is defined as any article that is "intentionally and verifiably false") has been studied widely in recent years, using context, word usage, and sources determine credibility [3]. This is very useful when dealing with long-form journalism, but isn't applicable for finding truth values of individual statements. Because a large part of this method involves analyzing the source the article came from, any lone statement from social media sites like Twitter or Facebook would be at a disadvantage. Alternatively, if attempting to fact-check a single sentence in an article, the truth value of this statement may not match up with that of the article as a whole, and could lead to inaccurate results.

Other research being done in this sphere, especially considering individual sentences, are essentially trying to get machines to use 'common sense.' Some of these models aim for the computer to 'understand' a given sentence, taking meaning from it and allowing a user to get the gist of a claim without needing to know exactly what was said. This can include determining the emotional connotation of a sentence, classifying opinions and reviews as negative or positive, or categorizing sentences based on broad subject matter. These models can start without prior knowledge of words and their meanings, only given an alphabet and large training set [4], or build off of individual words and their definitions in addition to grammatical rules [5], because they rely on previous experience. This means that simple, common-sense statements can be analyzed with relative ease, but statements that require more than just reasoning can not.

The most relevant research to the proposed model is about fact-checking and trying to determine the truth value of individual claims as described in *Using NLP for fact checking: a survey*. In this procedure, though, the algorithm must first determine whether the statement given is a claim at all. Because this seemed outside the scope of 5-week course and, "though research has gone into claim detection, there is no formal definition of what a claim is yet [2]," this step will not be included in the proposed model. Once a statement is confirmed to be a claim and is therefore able to be fact-checked, the algorithm searches through a fact database that stores prior claims that have already been manually fact-checked by experts. If the database does not have the information needed, the Internet can be utilized, in some situations just returning a snippet of the first search result to the user, and in other cases using sentiment analysis on the results to determine the credibility of the claim. Search results that include unassertive words can suggest a false claim, as well as

the use of first and second person pronouns as opposed to third person. When trying to fact-check a claim that is not included in a given fact database, "methods that utilize powerful search engines on the Internet perform best [2]."

3 Data

Since there isn't an open-source dataset of statements with truth values, I decided to create my own by scraping trivia websites and articles that tell the user whether a statement is true or not. When I have enough claims, ideally at least 500 instances, I would use an API to enter the statement in a Google search and scrape the results for 'snippets' to input into the dataset as well [6]. Once added to the dataset, cleaning will include removing markdown commands.

Using the bs4 package in python and multiple websites ([7], [8], [9], [10], [11], [12], [13]), I gathered 573 statements and their truth values. My original goal was 500 statements, but because I did not manually check for similarities, 573 gives a good margin of error in case there are duplicate statements from two different websites. Then I used an API to scrape Google and collect the "featured snippet" that comes up when the claim is searched, the descriptions from the first 10 results, and the number of results that come up in the search [14] [1]. This gave me a total of five features, four of type string ('statement', 'snippet', 'description', 'results') and one of type boolean ('truthValue').

4 Methods

I hope to classify these statements using a K nearest-neighbors algorithm. K nearest-neighbors (KNN) compares an instance to k other instances that are closest to it, aka its 'neighbors.' Because of KNN's simplicity and ease of use, this seems like a good option. Although KNNS can be slower than other algorithms and suffer from the curse of dimensionality (slowing down significantly as more features are considered), because I don't have a very large dataset or very many features to test, I think I will be fine.

A KNN model that could use non-categorical and non-numerical features was not included in scikit-learn's base packages, so with the help of the Towards Data Science website [15] I created my own algorithm for the purposes of natural language processing. The simplicity of KNNs was very useful here, because I otherwise may not have understood how to implement the process in my own code. This method does have the drawback of being slower than other models, especially considering that I relied on a naive solution of nested for loops when building many of the functions in the class.

5 Results

Using seven public websites with true/false questions, I scraped both the questions and their truth values into a dataset, giving me 573 instances. Once the questions and truth values were set in the dataset, using an API I scraped Google to give a summary of what the search for each question would bring up, the number of results, and the descriptions of the first ten sites in the search. A data dictionary is as follows:

- question: (type: string) An objective true or false statement. Ex. `Most people have 12 toes`
- truthValue: (type: boolean) A boolean value that corresponds to the question in its instance. Ex. 'most people have 12 toes' as a question would result in `FALSE` as a truthValue.
- snippet: (type: string) A one-sentence description that Google provides after searching the question in its instance. Ex. there will be no snippet that results from the question 'most people have 12 toes,' meaning the value in this instance will be `None`
- description: (type: string) A collection of the first 10 descriptions that show up in a Google search of the question. Ex. the description in this variable for the question 'most people have 12 toes,' begins with `Be it a baby born with 12 toes and 12 fingers or a girl with elf-like ears, some people are just so special, they stand out from the crowd without trying. Its almost as if Mother Nature was feeling too creative when she created them.` and carries on to have 9 other similar sentences
- results: (type: integer) The number of results that Google will come up with after searching the question. Ex. 'most people have 12 toes' will return `1772`— results.

I created my own user-friendly KNN algorithm (as seen in `model.py`) that used the nltk package to compare statements and store their similarities in an adjacency matrix. Using this matrix, I was able to predict truth values of the same

[1] Note that I used a different API for this initial scrape than I planned to use when constructing a web app to deploy the model. This is because this API allows for 1,000 searches per month which is convenient for populating a large dataset all at once, and the other API allows for 100 searches per day which is convenient for an app, where fewer searches will be made more often.

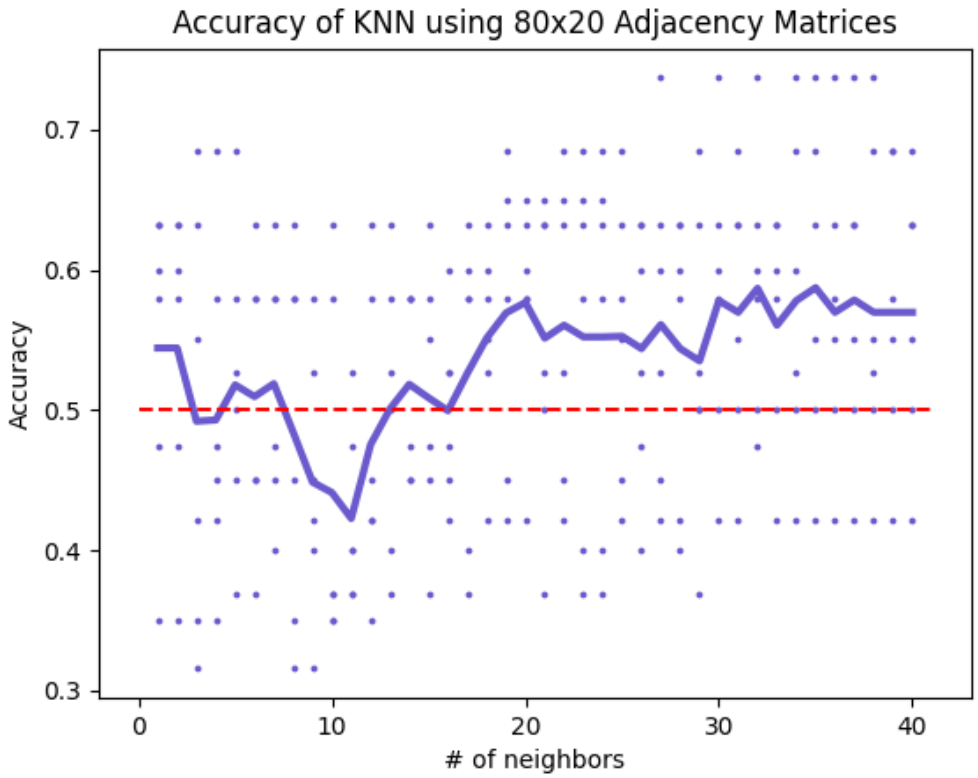

Figure 1: Scatter plot of the accuracy for models using k = 1-20 and plotting the mean of each k in purple. A red line at 50% shows the cut-off for the model being less-accurate than guessing at random.

statements using different values of k, allowing me to see the number of 'neighbors' that gave me the best results without repeating any calculations. I was able to achieve a mean accuracy of 58% at k = 20 that continued comparably all the way up to k = 40 (shown in figures 1 and 2) which I believe is sufficient considering the scope of this class and my access to computational resources.

With more computing power, I'm confident that I could increase the accuracy of this model by at least 15%. One reason that the accuracy is so comparatively low is because, in order to get results in a reasonable amount of time, I had to partition the data into 6 segments so as to create six 80x20 adjacency matrices (requiring knn.process() to compare 9,600 text statements) as opposed to one 480x120 matrix that would require 57,600 comparisons. Comparing 9,600 segments and saving the results took over two hours on my PC, so although the accuracy would be improved with a broader training set, attempting this would not have been realistic.

Once the adjacency matrix has been calculated, however, knn.predict() takes a very short time to output results since it uses the same matrix for all values of k, so long as the training and testing set stay constant. If I had access to a computer that could run for 12+ hours uninterrupted, I would be able to run knn.process() and then analyze my entire dataset at once, with the option of using even more data by scraping some additional websites.

References

[1] David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims, 2020.

[2] Eric Lazarski, Mahmood Al-Khassaweneh, and Cynthia Howard. Using NLP for fact checking: A survey. *Designs*, 5(3), 2021.

[3] Kai Shu and Huan Liu. *Detecting Fake News on Social Media*. Synthesis Lectures on Data Mining and Knowledge Discovery. Morgan & Claypool Publishers, 2019.

[4] Xiang Zhang and Yann LeCun. Text understanding from scratch. *CoRR*, abs/1502.01710, 2015.

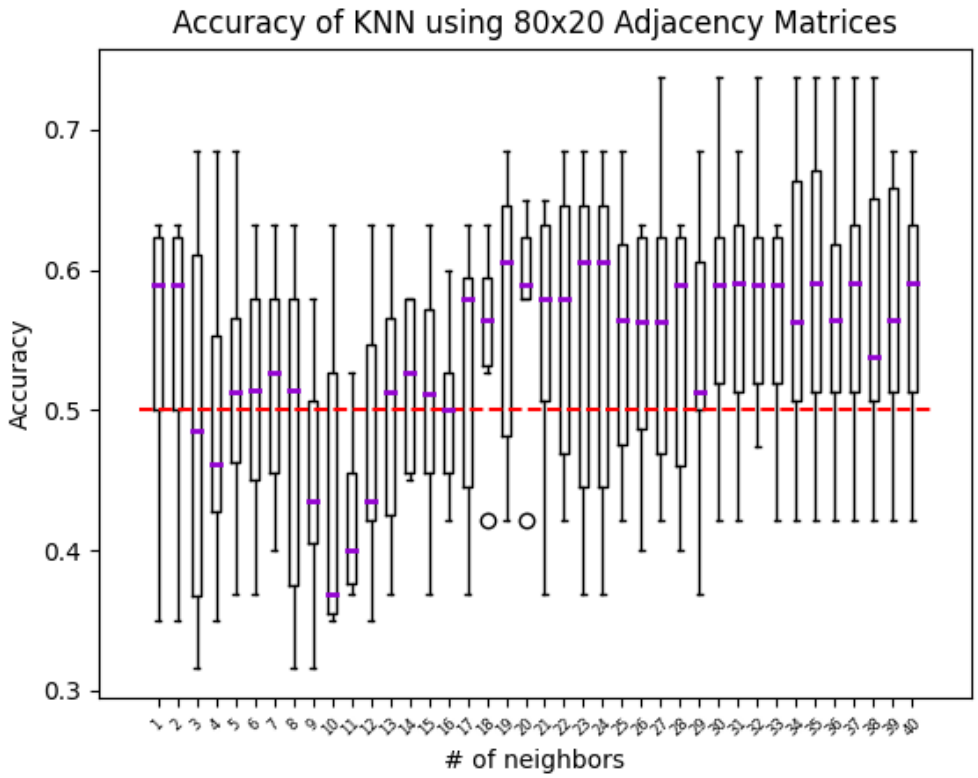

Figure 2: Box plot of the accuracy for models using k = 1-20 with the median shown as a purple line. A red line at 50% shows the cut-off for the model being less-accurate than guessing at random.

[5] Pei Wang. Experience-grounded semantics: a theory for intelligent systems. *Cognitive Systems Research*, 6(4):282–302, 2005.

[6] Roi Krakovski. Web search API. `https://rapidapi.com/user/contextualwebsearch`. Accessed: 2022-07-24.

[7] One hundred True or False questions. `https://www.flexiquiz.com/Help/inspiration/awesome-true-or-false-quiz-questions`. Accessed: 2022-07-26.

[8] 100 true or false questions. `https://www.signupgenius.com/groups/true-or-false-questions.cfm`. Accessed: 2022-07-26.

[9] True or false questions for kids. `https://www.playosmo.com/kids-learning/true-or-false-questions-for-kids/`. Accessed: 2022-07-29.

[10] 70 best true or false questions for an easier take on game night. `https://www.cosmopolitan.com/uk/worklife/a32612392/best-true-false-quiz-questions/`, 2022. Accesed: 2022-07-29.

[11] General knowledge: 45 true or false trivia questions & answers. `https://bigquizthing.com/trivia-questions-ans/general-knowledge-45-true-or-false-questions-and-answers/`. Accessed 2022-07-30.

[12] 100 awesome true or false questions for kids. `https://parenting.firstcry.com/articles/100-awesome-true-or-false-questions-for-kids/`, 2021. Accessed 2022-07-31.

[13] True or false questions. `https://ponly.com/true-or-false-questions/`, 2021. Accessed 2022-07-31.

[14] apigeek. Google search API. `https://rapidapi.com/apigeek/api/google-search3/`. Accessed: 2022-08-01.

[15] Text classification using k nearest neighbors. `https://towardsdatascience.com/text-classification-using-k-nearest-neighbors-46fa8a77acc5#:~:text=KNN%20algorithm%20is%20used%20to,to%20find%20the%20closest%20match.`, 2018.