Michael Schra
October 23, 2016

Given a Wikipedia article, classifies the article into one of two or more categories based on the article's text. For this example, the following two Wikipedia categories were chosen.

1) Theoretical physicists: https://en.wikipedia.org/wiki/Category:Theoretical_physicists
2) Superheroes appearing in films: https://en.wikipedia.org/wiki/Category:Superhero_film_characters

For example, give the article https://en.wikipedia.org/wiki/Richard_Feynman, the model should predict that the article belongs to a theoretical physicist. Similarly, given the article https://en.wikipedia.org/wiki/Professor_X, the model should predict the article belongs to a superhero.

Choices of Technologies

- Hardware: You may need 8GB of RAM. (I hit some memory errors until I cut the word corpus down to the 20% most common words.)
- I used Python 3.5 to complete the challenge as Python is an easy to code and understand programming language, and has numerous relevant packages available for it.
- Required Python Packages:
    - requests Used for its API and JSON functions
    - pickle Used to save the Wikipedia data locally, due to the long call times
    - re Used for text manipulation and pre-processing
    - random Used to randomize the train/test sets
    - nltk Used for text pre-processing and document classification

Obtaining/Wrangling/Cleaning Data – getWikiData.py

When this model is first imported it attempts to download a list of stop words (common English words to remove from the article text) from nltk.corpus if it is not found locally. After the first run, it should just load them from cache.

FUNCTION: load_data(classes)

Attempts to load the data from the local cache (via the pickle package) if there is an error, or it's not found, it calls **get_Wiki_Data(classes)** to download the data and return it. The default is to call this function from **main.py** when the program is loaded, instead of **get_Wiki_Data(classes),** unless the **refresh_data** flag is set to True.

FUNCTION: save_data(articles)

Saves the dataset, **articles,** locally (via the pickle package) so that the next time **main.py** is ran, it can just be loaded by the **load_data(classes)** function.

FUNCTION: get_Wiki_Data(classes)

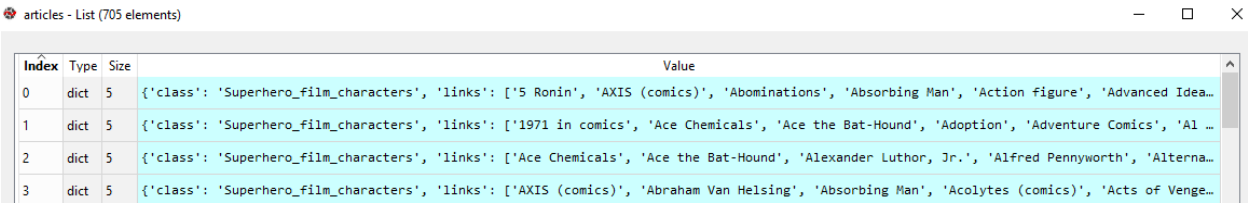
Loops through each provided category (the code can handle more than two classes) and calls the **getArticles(category)** function to get a list of articles and their data in a given Wikipedia category, appending the data into a list called **articles**. Afterwards it calls the function **save_data(articles)** to save the data locally (via the pickle package) because of the long amount of time it takes to download all the Wikipedia data due to having to make two API calls for each Wikipedia article in each category.

FUNCTION: getArticles(category)

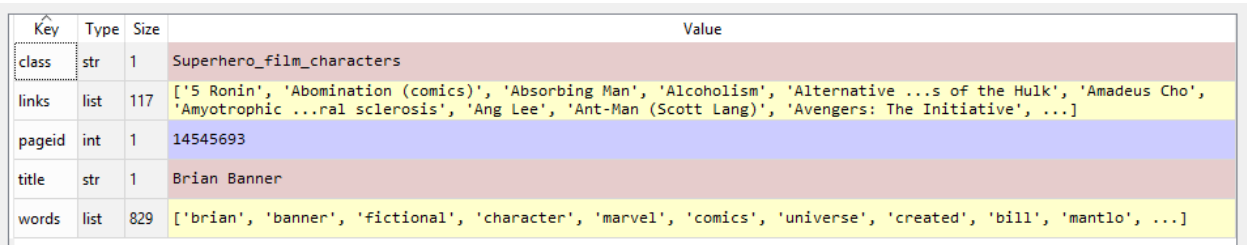
This is the first step in obtaining the data from the Wikipedia API. Given a category, it constructs, then makes a call to the *categorymembers* API to obtain a list of all the articles. This is complicated by the fact that the API restricts results to the first 500. Therefore, a loop has to be used until the API no long returns a 'API Call Continuation Key'

Once it has the list of all the articles, it builds a dictionary for each one with the variables (class, title, pageid, links, and words). For the last two, it has to call **getLinks(pageid)** and **getWords(pageid)** respectively.

Finally, it returns a list of dictionaries, called **articles**.



Below is an example of the resulting dictionary data structure for one of the list elements.



FUNCTION: getLinks(pageid)

Using the same API continuation logic as **getArticles()** it returns a list of all the Wikipedia articles the provided pageid links to.

FUNCTION: getWords(pageid)

Using the same API continuation logic as **getArticles()** it first returns a list of all words from the Wikipedia article the provided pageid links to. The following text processing is also done:

- Replaces newline characters with a space (so the words can later be tokenized on the space character)
- Removes everything not a character or space, such as punctuation and numbers.
- Removes back to back spaces which could be created by the above logic
- Converts all the text to lower case, as Python string matching is case sensitive.
- Splits the string of words to a list where every element is a word
- Using NTLK's list of English stop words, removes common words from the list (such as 'the')
- Originally I stemmed the words, but it made the model worse. My intuition is that nouns are much more important than verbs in this classification task, and stemming can have undesirable side effects.

Finally, the function returns the list of processed words.

FUNCTION: getPageID(title)

This function is only used to process a user provided Wikipedia URL's title (everything after the last '/' in the URL) to retrieve the pageid so that **getLinks(pageid)** and **getWords(pageid)** can be called and the document classification models ran on the resulting data.

Machine Learning – getClassifierModels.py

Naïve Bayes Classifier from Python's NLTK package felt like a good first choice because it is easy to learn, trains a model fast, can handle multiple classes, and is surprisingly accurate despite its name. After receiving accuracy rates ~95% on the test set, I decided to stick with it and see if I could tune it to be a little more accurate.

For feature selection, I tried different combinations of word frequencies, word occurrences (T/F: frequency > 0), and "Links to" (T/F: whether the article links to another article) as Wikipedia is a massive linked list. I first found that word frequencies and occurrences were equally accurate, but word frequencies added a lot of overhead to the model because a feature needed to be created for every possible combination of a word and # of occurrences in an article. Therefore, I dropped word frequencies in favor of just using word occurrences.

I also found that combining word occurrences and "Links to" did not increase the accuracy by any significance compared to either model alone. Therefore, I decided to test to separate NBCs – one for word occurrences and one for 'Links to'. (Please see the Future Work section for additional reasoning)

FUNCTION: getLinks(articles)

Returns a list of all the unique Wikipedia links in the corpus.

FUNCTION: getCommonWords(articles)

Returns a list of all the top 20% most common unique words in the corpus. This comes out to include only words which have at least 10 occurrences. Any less and they won't be valuable predictors.

FUNCTION: getFeatures (article_items, all_items, feature)

Returns a feature set, which is a dictionary of Boolean values for whether the article's items (article_items) appear in the corpus (all_items). This function works for both words and links using the parameter *feature*.

FUNCTION: getDataSet_words (articles, common_words):

Given the list of all the Wikipedia articles, and the output of **getLinks(articles),** calls **getFeatures()** for every article and returns the data in the format required to build a Naïve Bayes Classification model.

FUNCTION: getDataSet_links(articles, all_links)

Given the list of all the Wikipedia articles, and the output of **getCommonWords(articles)**, calls **getFeatures()** for every article and returns the data in the format required to build a Naïve Bayes Classification model.

FUNCTION: getNaiveBayesClassifier(data)

Given the data from getDataSet_words() or getDataSet_link() trains and returns a Naïve Bayes Classifier model.

Statistics and Testing – main.py (commented out)

My test code is contained in main.py, but commented out, so as to not interfere with its user interface functionality. Using a Train set ratio of 0.75, I received the following accuracies and Most Informative Features for the two models on the test set.

```
Word Classifier accuracy percent: 95.1%
Most Informative Features
        Contains(voiced) = True           Superh : Theore =    190.1 : 1.0
       Contains(returns) = True           Superh : Theore =    183.3 : 1.0
      Contains(animated) = True           Superh : Theore =    181.5 : 1.0
       Contains(villain) = True           Superh : Theore =    150.6 : 1.0
       Contains(decides) = True           Superh : Theore =    119.6 : 1.0
         Contains(squad) = True           Superh : Theore =    117.9 : 1.0
    Contains(continuity) = True           Superh : Theore =    111.0 : 1.0
       Contains(injured) = True           Superh : Theore =    111.0 : 1.0
          Contains(shot) = True           Superh : Theore =    109.3 : 1.0
        Contains(heroes) = True           Superh : Theore =    106.9 : 1.0

Link Classifier accuracy percent: 97.2%
Most Informative Features
       LinksTo(Iron Man) = True           Superh : Theore =     96.5 : 1.0
LinksTo(Magneto (comic…) = True           Superh : Theore =     47.7 : 1.0
LinksTo(Library of Con.) = True           Theore : Superh =     44.3 : 1.0
    LinksTo(Doctor Doom) = True           Superh : Theore =     43.7 : 1.0
           LinksTo(Wiki) = True           Superh : Theore =     27.4 : 1.0
         LinksTo(Genius) = True           Superh : Theore =     27.4 : 1.0
       LinksTo(Facebook) = True           Superh : Theore =     23.4 : 1.0
  LinksTo(Graphic novel) = True           Superh : Theore =     21.3 : 1.0
      LinksTo(Physicist) = True           Theore : Superh =     21.1 : 1.0
LinksTo(First appearanc) = True           Superh : Theore =     20.1 : 1.0
```

The Classifier that used “Links To” as a feature was slightly more accurate, 97% to 95.1%, however with these small sample sizes it is not a significant difference to say one model is better than the other.

Surprisingly, all of the Top 10 Most Informative Features for the word classifier are Superhero related (True -> Article is Superhero, False -> Article is Phsycist). My best guess as to the reason why is the training set of Superhero articles was only 1/3 that of the training set for physicist articles, and Superhero articles are probably written using a lot of the same terms (voiced, villian, animated, etc.) whereas articles about Theoretical Physicists probably vary more in their writing style and word choice.

Confusion Matrices

I wrote a function to generate a confusion matrix, after an unsuccessful search for one built into the NTLK package. I ran it across the entire dataset, because the train/test sets were so small**,** to examine which articles were misclassified.

Word Classifier Confusion Matrix

	Actual	
Prediction	Superhero_film_characters	Theoretical_physicists
Superhero_film_characters	**155**	14
Theoretical_physicists	26	**510**
Total:	181	524
Accuracy:	**85.6%**	**97.3%**

The overall **accuracy was 94.3%.** It is clear that the model is much more accurate at classifying theoretical physicists (possibly due to the larger training set). A quick glance through the articles that were misclassified didn't reveal many commonalities, except that the misclassified theoretical physicists tended to be more well-known (I knew who most of them were) and therefore films have likely been made about them, which were discussed in their Wikipedia article.

Link Classifier Confusion Matrix

	Actual	
Prediction	Superhero_film_characters	Theoretical_physicists
Superhero_film_characters	**168**	0
Theoretical_physicists	15	**522**
Total:	183	522
Accuracy:	**91.8%**	**100%**

The overall **accuracy was 97.9%.** Again, it is clear that the model is much more accurate at classifying theoretical physicists (possibly due to the larger training set). A quick glance through the articles that were misclassified didn't reveal any commonalities.

Future Work

Given more time, I'd like to rerun the project down sampling the classes to generate classes of equal amounts of training data as the models seems to be slightly biased towards predicting Theoretical_physicists.

Also, because Wikipedia is one massive collection of linked articles, I'd love to build a linked list that looks at every article the articles in the category link to (many of them link to each other), then looks at every article those articles link to until a fully connected graph is constructed.

While this probably still wouldn't be 100% accurate, it could help in the situations where the article you want to classify has very little content. Some examples:

- A superhero article only links to an uncommon publisher, which links to Marvel Comics, which links to many articles that other superhero articles link to.
- A physicist's article only links to an uncommon topic in theoretical physics, and that topic links to other theoretical physicists in the theoretical physicist's class.

The unit of measure could be the class's minimum or average of the minimum number of degrees to every member of each class.

Source: https://en.wikipedia.org/wiki/Wikipedia:Six_degrees_of_Wikipedia

Usage – main.py

The first component of **main.py** is the developer variables listed below. As mentioned before, the classes can be modified, or added to, but the Wikipedia data must then be refreshed.

```
#Developer Variables
classes = ['Superhero_film_characters', 'Theoretical_physicists']
refresh_data = False        #Set to True to refresh the Wikipedia data
```

The program then either downloads and processes the Wikipedia data, or loads it from local cache via the pickle package through **getWikiData.py**. Afterwards, it calls functions within **getClassifierModels.py** to build and return classification models. Finally, it continually prompts the user for a Wikipedia URL and predicts the class (until the user chooses to exit). An example test run is provided below:

Some Interesting Results

Wikipedia URL	Actual	In Categories	Word Classifier	Link Classifier
https://en.wikipedia.org/wiki/Professor_X	Superhero	Yes - Correct	Correct	Correct
https://en.wikipedia.org/wiki/Michio_Kaku	Physicist	Yes - Correct	Correct	Correct
https://en.wikipedia.org/wiki/Richard_Feynman	Physicist	Yes - Correct	Incorrect*	Correct
https://en.wikipedia.org/wiki/Aquaman	Superhero	No - Correct	Correct	Correct
https://en.wikipedia.org/wiki/Victor_Weisskopf (Not in training data – took a while to find one)	Physicist	No - Correct	Correct	Correct
https://en.wikipedia.org/wiki/Gordon_Freeman (Video game hero theoretical physicist)	Both	No - Correct	Superhero	Physicist
https://en.wikipedia.org/wiki/Neil_deGrasse_Tyson (The world's most charismatic physicist)	Media Star / Physicist	No - Correct	Superhero*	Physicist
https://en.wikipedia.org/wiki/Professor_Farnsworth (TV Character mad scientist – specializing in physics)	TV Character / Physicist	No - Correct	Physicist	Physicist

*As discussed earlier, the few misclassified theoretical physicists seemed to be the ones with lots of media exposure, such as films, including Einstein and Hawking and thus would have a lot of words relating to film/media in their articles.

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Michael>cd desktop

C:\Users\Michael\Desktop>python main.py
Loading and cleaning the Wikipedia Data. This may take a few minutes.
The Wikipedia data can't be loaded from the local cache. It is being downloaded again, please be patience.

50 Superhero_film_characters articles are loaded.
100 Superhero_film_characters articles are loaded.
150 Superhero_film_characters articles are loaded.
All Superhero_film_characters articles downloaded!

50 Theoretical_physicists articles are loaded.
100 Theoretical_physicists articles are loaded.
150 Theoretical_physicists articles are loaded.
200 Theoretical_physicists articles are loaded.
250 Theoretical_physicists articles are loaded.
300 Theoretical_physicists articles are loaded.
350 Theoretical_physicists articles are loaded.
400 Theoretical_physicists articles are loaded.
450 Theoretical_physicists articles are loaded.
500 Theoretical_physicists articles are loaded.
All Theoretical_physicists articles downloaded!

Creating the article classification models. This will take a few minutes...

Enter a Wikipedia article's URL for classification (quit to exit):
https://en.wikipedia.org/wiki/Professor_X

This Wikipedia article is in the Superhero_film_characters category.
The Word Classifier Model predicts the Superhero_film_characters category.
The Link Classifier Model predicts the Superhero_film_characters category.

Enter a Wikipedia article's URL for classification (quit to exit):
https://en.wikipedia.org/wiki/Michio_Kaku

This Wikipedia article is in the Theoretical_physicists category.
The Word Classifier Model predicts the Theoretical_physicists category.
The Link Classifier Model predicts the Theoretical_physicists category.

Enter a Wikipedia article's URL for classification (quit to exit):
https://en.wikipedia.org/wiki/Richard_Feynman

This Wikipedia article is in the Theoretical_physicists category.
The Word Classifier Model predicts the Superhero_film_characters category.
The Link Classifier Model predicts the Theoretical_physicists category.

Enter a Wikipedia article's URL for classification (quit to exit):
https://en.wikipedia.org/wiki/Aquaman

This Wikipedia article is not in either of the categories.
The Word Classifier Model predicts the Superhero_film_characters category.
The Link Classifier Model predicts the Superhero_film_characters category.

Enter a Wikipedia article's URL for classification (quit to exit):
https://en.wikipedia.org/wiki/Victor_Weisskopf

This Wikipedia article is not in either of the categories.
The Word Classifier Model predicts the Theoretical_physicists category.
The Link Classifier Model predicts the Theoretical_physicists category.
```

```
Enter a Wikipedia article's URL for classification (quit to exit):
https://en.wikipedia.org/wiki/Gordon_Freeman

This Wikipedia article is not in either of the categories.
The Word Classifier Model predicts the Superhero_film_characters category.
The Link Classifier Model predicts the Theoretical_physicists category.

Enter a Wikipedia article's URL for classification (quit to exit):
https://en.wikipedia.org/wiki/Neil_deGrasse_Tyson

This Wikipedia article is not in either of the categories.
The Word Classifier Model predicts the Superhero_film_characters category.
The Link Classifier Model predicts the Theoretical_physicists category.

Enter a Wikipedia article's URL for classification (quit to exit):
https://en.wikipedia.org/wiki/Professor_Farnsworth

This Wikipedia article is not in either of the categories.
The Word Classifier Model predicts the Theoretical_physicists category.
The Link Classifier Model predicts the Theoretical_physicists category.

Enter a Wikipedia article's URL for classification (quit to exit): quit

C:\Users\Michael\Desktop>
```