MOVIE RECOMMENDATION WITH NATURAL LANGUAGE PROCESSING

BY GIDEON BLINICK

BACKGROUND

- The problem I am solving is how can I provide a user with movies they want to see, assuming that I am provided with a movie that I know the user already knows and likes?
- The client is any user that likes movies and wants to see more movies that they like.
- The motivation for this problem is my own experience. I enjoy a good movie now and then but it is hard to know what to see. Using rating websites is not always a reliable way of finding great movies. An approach that takes into account my preferences is optimal.

FORMULATION OF THE PROBLEM AS A DATA SCIENCE PROBLEM

- Movie Recommendation is, naturally, a recommendation problem.
- Recommendation problems fall under neither Supervised Learning nor Unsupervised Learning, but rather are **information retrieval** problems [1].
- There are generally 2 types of recommender systems:
  - Content-based: make recommendations based on the content of an item/thing itself (i.e. you like item A, so you'll probably like item B which is very similar to item A).
  - Collaborative-based: make recommendations based on a user profile (i.e. your profile is similar to person A, and I know he likes bananas, so I will recommend bananas to you).
- My recommender systems are content-based. A user provides a movie and the recommendations returned are entirely based on the provided movie, not the users' profile.

DATASET DESCRIPTION

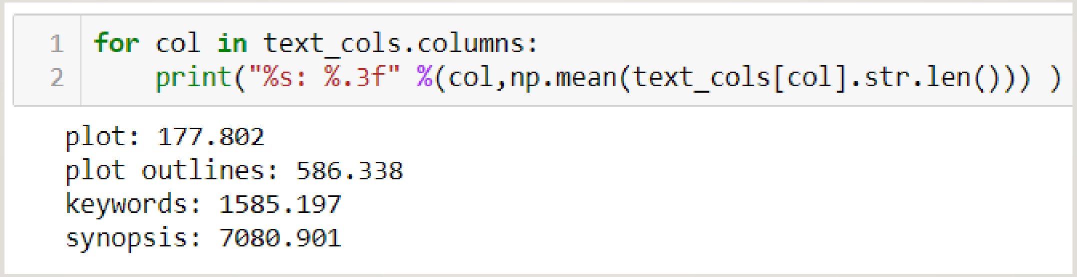
- I assembled my own dataset from IMDb data.
- The first data source I used was IMDb's public datasets. There is a lot of data there, but unfortunately no text data. So I needed another way of obtaining text data describing movies.
- That's when I found the IMDbPY API. It allowed me to pull plot summaries and plot keywords directly from IMDb.
- I used the public datasets to get the top 10,000 movies by number of reviews. I then pulled the plot summary and plot keywords for those movies from IMDb.

DATA WRANGLING

- There wasn’t too much to do in this stage.
- All the tables from the public datasets that I used had a movie ID column that made joining them very easy.
- Furthermore, when I pulled text data via the API, I simply used the movie IDs that I already had.

EXPLORATORY DATA ANALYSIS

- First, I looked at the average length (by number of characters) for each of the 4 text columns that I pulled: plot, plot outline, keywords, and synopsis.



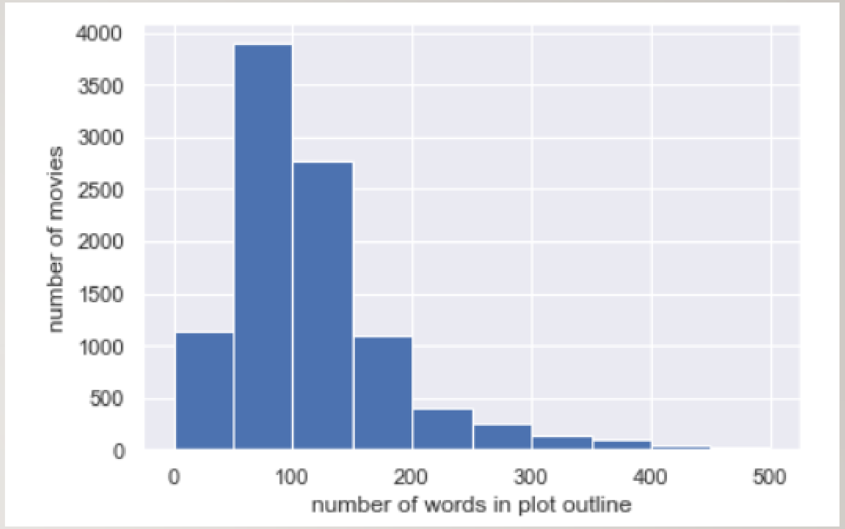
```
for col in text_cols.columns:
    print("%s: %.3f" %(col,np.mean(text_cols[col].str.len())) )
```

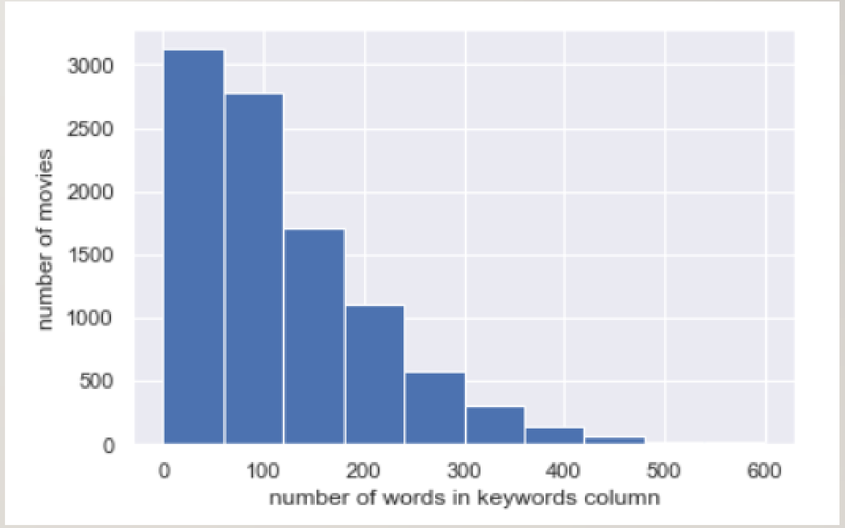
```
plot: 177.802
plot outlines: 586.338
keywords: 1585.197
synopsis: 7080.901
```

- I found that synopsis contained the most characters, followed by keywords, followed by plot outlines, followed by plot.
- At this point, I made the decision to just analyze plot outlines and keywords. I chose plot outlines because it wasn't too long or short and keywords because it had the richest info.

EXPLORATORY DATA ANALYSIS

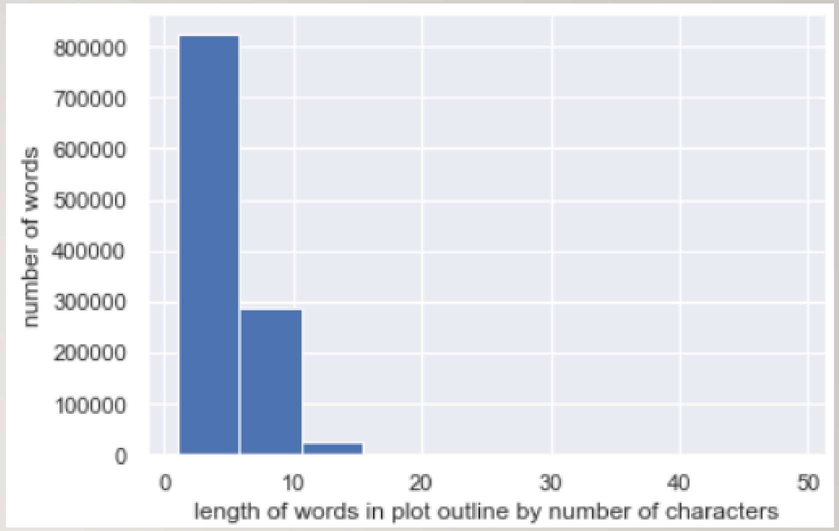
- Next, I looked at the distributions of the number of words for keywords and plot outlines.
- When looking at the summary statistics for these distributions, I found that the mean number of words for keywords is only slightly greater than the number of words for plot outline (~122 vs. ~115).
- This is misleading however, since most keywords are compound words (e.g. "dc-comics").

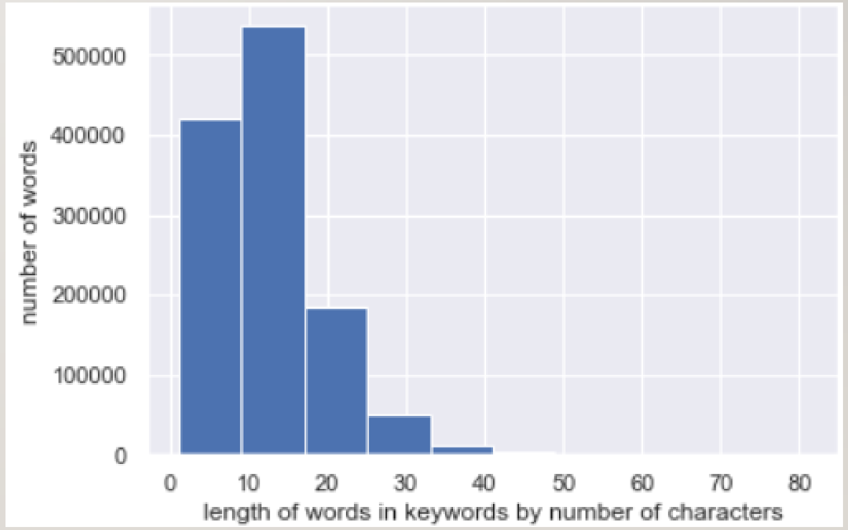




EXPLORATORY DATA ANALYSIS

- When I plotted the **lengths** of words for plot outlines and keywords, I obtained a result that shows the difference between the 2 columns.
- The length of words in the keywords column is far greater, on average, than the length of words in the plot outlines column.
- This makes sense. Keywords by their nature are meant to be descriptive ('information rich') and such words are longer.





MODELLING

- I used keywords for the model based on finding that they were best from EDA.
- Most of the normal preprocessing steps for text did not need to be performed:
  - Tokenization: keywords already comma-separated
  - Lowercasing: already lowercased
  - Removing stop words and punctuation: keywords do not have either so no need.
  - Lemmatizing: this is something I considered doing. I decided not to after seeing how the meaning changed for a variety of lemmatized words. For example, "avengers" (the superhero team) became "avenger". In my opinon, more information is lost than gained by lemmatizing in this case.
- I created 3 different recommender models: a Tf-idf model with cosine similarity, a Jaccard recommender model, and a word-count cosine similarity model

1ST MODEL: TF-IDF WITH COSINE SIMILARITY

- Tf-idf stands for *term frequency-inverse document frequency* and is a way of assigning importance to words in a document based on how often they appear in that document and in the other documents in a corpus (here corpus just means 'collection of documents'). Tf-idf allows you to get a sense of what a document is about by seeing what words are most important in it.
- I created the model using Gensim (using the Dictionary and TfidfModel classes)
- Cosine similarity works by measuring the cosine of the angle between 2 vectors. In my case, the vectors I was using were each movie's keywords representations in my tf-idf model.

1ST MODEL: TF-IDF WITH COSINE SIMILARITY

- The steps this model takes when recommending movies then is as follows:
  1. Retrieve the movie's keywords;
  2. Convert those keywords into a bag-of-words model;
  3. Convert the bag-of-words model into a tf-idf representation of those words;
  4. Find the n most similar movies to the movie's representation by using cosine similarity to calculate the similarity between every movie and the given movie and return the top n results.
- Note that the model can be used to find the most similar movies for any given set of keywords.

2ND MODEL: JACCARD SIMILARITY

- Jaccard similarity is defined as the intersection of 2 sets divided by the union of those sets. So this model finds the similarity between 2 movies by dividing the number of common keywords between 2 movies by the number of unique keywords in the union of the movies' keywords.
- Mathematically, this is given as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Jaccard Similarity (image taken from Wikipedia)

- Note that there is no need for document vectors here as there was for finding cosine similarity.

3RD MODEL: COSINE SIMILARITY BETWEEN WORD COUNTS

- My last model is similar to my first model in that it calculates cosine similarity between 2 movie vectors.
- However, unlike the first model, here each movie vector is defined by simple word counts (that is, the number of times each word appears for a given movie) rather than tf-idf.
- This alternate implementation is performed with the CountVectorizer class in scikit-learn which converts a collection of text documents (i.e. keyword lists in this case) into a matrix of token counts.

DISCUSSION AND NEXT STEPS

- When using the models with a few movies with which I am familiar, I get recommendations that are good matches.
- A good example to use is Marvel Studios' 2012 movie "The Avengers".
- Here are the top 5 results I get using each model:
    - 1st Model: Avengers: Age of Ultron, Avengers: Infinity War, Iron Man 2, Captain America: Civil War, Captain America: The Winter Soldier
    - 2nd Model: Avengers: Age of Ultron, Avengers: Infinity War, Captain America: The Winter Soldier, Captain America: Civil War, Thor: The Dark World
    - 3rd Model: Avengers: Infinity War, Avengers: Age of Ultron, Captain America: Civil War, Iron Man 2, Justice League

DISCUSSION AND NEXT STEPS

- The most obvious problem with the models as they currently exist is they do not have evaluation metrics.
- To be fair, this is something of an issue for all recommender systems.
- My first next step then would be to come up with evaluation criteria by which I could compare the models and by which I could improve them.
- The next thing I would do after that is put the models into production. This follows general machine learning best practice of putting simple models into production as fast as possible.
- I would use Flask for deployment.

DISCUSSION AND NEXT STEPS

- My 3rd improvement would be to incorporate non-text features into the model to use for recommendation.
- IMDb has many features in their public datasets and available via IMDbPY and I was initially going to use 4 numeric ones (year of release, runtime, number of reviews, and rating) as well as what genre the movie belongs to (a categorical feature).
- A 4th improvement would be clustering the movies for better insight into the data (better EDA).
- A 5th improvement would be creating additional models that use other notions of similarity and seeing how they compare across various evaluation criteria.

REFERENCES

- [1] https://www.quora.com/Where-do-recommender-systems-fall-in-machine-learning-approaches
- Reference Material on Cosine Similarity with a td-idf Model: https://www.oreilly.com/learning/how-do-i-compare-document-similarity-using-python
- Reference Material on Jaccard and Cosine Similarity Models with Text Data: https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50