Cinema Information Retrieval from Wikipedia

John Ryan
Courant Institute of Mathematical Sciences
251 Mercer St.
New York, NY 10012
`jpr349@cims.nyu.edu`

Abstract

In this paper, we explore various methods for information retrieval from a text corpus and how they compare when used to find the common link between several elements of a query. We do so with several experiments on a corpus of Wikipedia articles on movies and actors in cinema.

1 Introduction

Perhaps the best way to reference a movie, having forgotten the title, is to hint at it by listing starring actors. ”I can’t remember the exact name, but it’s the one with Jennifer Lawrence and Bradley Cooper.” Conversely, for lack of a name to place to a face, one may say ”she was in Annie Hall and The Godfather,” and hope that someone else can name the actress. Nowadays, this problem is quickly solved by Google - for example, a Google search of ”Annie Hall The Godfather” returns the IMDB page and the Wikipedia page for Diane Keaton within the first three results. The search has found the common link between the two different entities of the same category which we inputted.

Suppose we’d like to perform this task offline on a corpus with an algorithm of our own. Jurafsky and Martin (2000) suggest using a TF-IDF vector space model and evaluating relevancy by cosine similarity. We will implement this system, and we will compare the results when we change the term weighting, and when we change the pairwise similarity metric.

2 Problem Statement

Our corpus is about 4,300 Wikipedia articles (coverted to ASCII-only .txt documents). Around twenty percent of these articles describe actors, and the rest describe movies. The goal of our system will be to take as input a query of several elements of the same category (in this case, either actors or movies) and output the common link between those elements in a different category. For example, the system should respond to the input ”Annie Hall The Godfather” with ”Diane Keaton,” and the input ”Jennifer Lawrence Bradley Cooper” with ”Silver Linings Playbook” (or ”American Hustle”).

2.1 Corpus

The corpus contains about 4,300 documents; 1,000 correspond to actors and 3,300 to movies. The documents are plaintext files obtained by downloading the Wikipedia articles’ .html files and using the textutil tool in the Mac OS X terminal to convert from .html to .txt. Since the Vectorizer requires ASCII characters, a Java program using the Normalizer class was used to reduce non-ASCII which corresponded with ASCII characters (so that, for example, ”Renée Zellweger” would become ”Renee Zellweger” instead of ”Rene Zellweger”), and to get rid of all other non-ASCII symbols. Finally, everything below the ”References” section header in the article (every Wikipedia page has one) was deleted to save space and keep things clean.

2.2 Vectorizer

Since the system will evaluate relevancy by comparing vectors by some pairwise metric, our first task is to decide how to represent queries and documents as vectors with real entries. In this case, we will consider the entries of the vectors (i.e. the tokens) to be *n*-grams in the vocabulary of the corpus.

The clear first choice for a vector representation is contained within sklearn's "feature extraction" library, and it is found by the CountVectorizer. In this model, the value for a given token in the vector of a query or document is just its frequency. For example, if we are using CountVectorize with $n = 1$, then the vector for "foo foo bar bar bar" would look like $\{2, 3\}$ (where remaining entries for other tokens in the corpus would be 0).

Our second choice for vector representation is analogous to our first, and it is found by the HashingVectorizer. In this model, the entry for a given n-gram is again its frequency in the document, but now it is computed using a hash table. How is this different? On the one hand, our computations are sped up, since determining if we have seen a certain token before takes constant time now. However, we risk having two tokens map to the same entry (a collision in the hash table), and having the vector be slightly misrepresentative. According to the documentation given by sklearn, this is not a big problem in general; however, we will see that our system's success may be affected by these collisions, undoubtedly owing to the size of our corpus.

The third and final choice for vector representation which we will test in our implementation is that of the TfidfVectorizer, The key difference in this model is that terms which appear only in a few documents are given much more weight. We expect this to be a much more effective model than the other two, as it is the only one we will use which weights a term with respect to both its document and the corpus. An example of the power of this weighting is found by considering the query "Ellen Page Marion Cotillard." Whereas the CountVectorizer and HashingVectorizer would mistakenly assign more relevancy to documents containing multiple occurrences of "page" and articles for other actresses named Ellen, the TfidfVectorizer would recognize how "page" and "ellen" are much more common in the corpus than "marion" or "cotillard," and thus would assign relevancy more strongly to those containing the latter terms. We expect that this impact will also be notable in our data when we include in our input a movie with the token "american" within, such as "American Hustle."

One might argue that cases such as those described in the previous paragraph will be fixed when n is set to 2. For example, the system wouldn't fall into the trap of considering "Ellen DeGeneres" when the token is "Ellen Page" instead of "Ellen". However, suppose we use a query such as "John Ratzenberger Billy Crystal" when looking for a certain Pixar film. When $n = 2$, we are saved from assigning relevancy to all the John's and Billy's in the corpus; however, as far as Pixar movies are concerned, the token "John Ratzenberger" is redundant, since he has been in every single Pixar movie. It would help if, recognizing that Billy Crystal is more helpful in distinguishing the movie, his name be given more weight.

2.3 Comparing Vectors: Pairwise Metrics

In our implementation, we will compare results from two different metrics for pairwise similarity of vectors. The motivation for such techniques is that we'd like a quick way to decide how similar a document is to a query by analyzing the difference between the two vector representations in the multi-dimensional vector space.

The Cosine Similarity metric is solely concerned with the angle between the two vectors. We know that, for vectors x and y,

$$xy^T = |x||y| \cos \theta$$

where θ is the angle between the two vectors. Thus, to find where the angle is small, we only need to find where

$$\frac{xy^T}{|x||y|} \tag{1}$$

```
What is the query?
steve martin,
john candy,
laila robins
Calculating Tfidf Matrix
Using TFIDF with Cosine Similarity
10 Planes,_Trains_and_Automobiles
9 Steve_Martin
8 Nothing_but_Trouble_(1991_film)
7 Father_of_the_Bride_(1991_film)
...
```

Figure 1: Sample output

is close to 1 (this is algorithmically easy). Thus, if x and y plugged into (1) gives 0.5 and x and y' gives 0.75, then we believe that document y' is more relevant to document x than document y. We note that if x and y are normalized to begin with, than (1) is equivalent to

$$xy^T \tag{2}$$

This leads us to the second pairwise metric we will use to evaluate query/document relevance: the Sigmoid Kernel. Starting with normalized x and y, the Sigmoid Kernel is

$$\tanh\left(\gamma xy^T + c_0\right)$$

where γ is known as the slope and c_0 as the intercept (these are chosen for us by sklearn). Again, we are looking for x and y which make this as close to one as possible.

The Sigmoid Kernel is included in the experiments detailed in this paper and in the final program because it produced the most interesting results of the pairwise metrics provided by sklearn.

3 Experiments

Since we will be working with two different pairwise metrics and three different vector space models, we have six different techniques by which to extract information from the corpus so as to answer a user's query. Keeping in mind that the goal is to identify the common link among elements of the user's query, we will ask random students at New York University for sample queries along with the intended common link. The reason for picking random students (as opposed to other computer science students) is to attempt to best emulate the average Google user and his/her lack of understanding of the underlying algorithms and what queries will make it harder for the system. With at least 40 such queries, we will begin testing.

The test works by having a user pass the system a query, to which the system responds with ten guesses as to the common link between the elements in the query. For simplicity, we consider sequels containing the same cast to be identical to originals (for example, a guess of "The Matrix Revolutions" is as good a guess as "The Matrix" when given a query containing actors in both movies). A sample output is given in Figure 1.

Because we are intentionally asking the system for at least 9 incorrect results, not much is found by evaluating F-score in the usual manner. Instead, we will evaluate the different techniques by awarding points for each correct guess based on its position in the output. For example, in Figure 1, the technique earned itself 10 points because the user was thinking of "Planes, Trains, and Automobiles." If the user had been thinking of "Father of the Bride," that would have been worth 7 points. If the user's intended link doesn't appear on the list, that is worth 0 points. After the 40 rounds of testing, the points will be summed up and compared.

3.1 Results

The experiments were successful, and the results match what we predicted in some cases, and surprise us in others. Before discussing the performances of the information retrieval techniques, we make some comments on the execution of the experiments.

The first aspect worthy of comment is the runtime. Calculating the matrices for the vectors took about 11 seconds for $n = 1$, regardless of the vectorization method. Furthermore, when $n = 2$, the CountVectorizer and

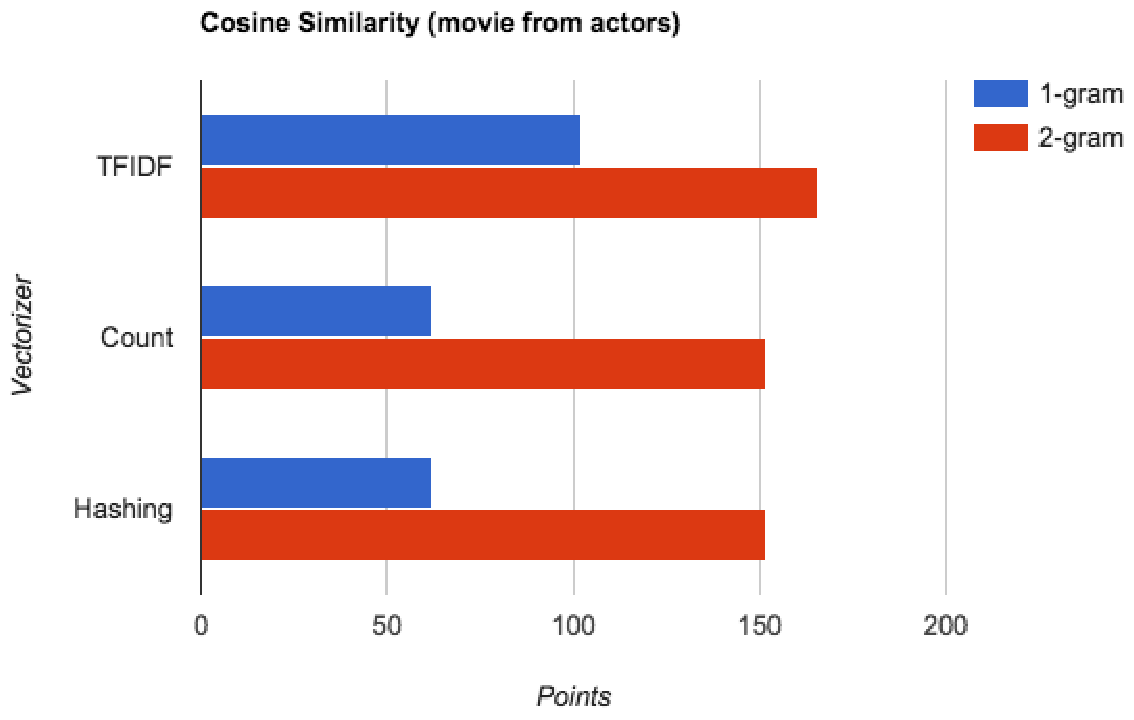


Figure 2: Results for the six techniques

TfidfVectorizer took 55 seconds and the HashingVectorizer took 25 seconds. It must be the case that an effective search engine would preprocess these values when it crawls the web for pages to return. Then, the pairwise metric calculations and subsequent ranking of the vectors took less than a second, a speed surely owing to the algorithmic efficiency mentioned earlier.

Figure 2 contains the points data for using Cosine Similarity as the pairwise metric when trying to find the movie in which all actors in the query appeared. The maximum possible score would be 200, and would have been achieved if the intended movie had been the system's first guess for all 20 rounds.

Many aspects of these results were expected. In both the 1-gram and 2-gram cases, using TFIDF for the vector representations resulted in the most success. Furthermore, the scores didn't vary between using CountVectorizer and HashingVectorizer. However, something quite unpredictable happened during the testing that the graph in Figure 2 doesn't represent. In many cases where the other five techniques failed, using Sigmoid Kernel with CountVectorizer succeeded greatly. One such example (drawn from the system's output files, all of which are included in the package corresponding to this paper) is when a user queried "daniel craig eva green mads mikkelsen," referring to the movie "Casino Royale." To that input, the system was able to correctly identify the movie for the other five techniques, earning 8 points for each of those methods. However, the Sigmoid Kernel technique was wrong in all of its top ten guesses, earning 0 points.

On the other hand, when passed the query "marion cotillard joseph gordon-levitt ellen page" (referring to "Inception"), whereas the TFIDF techniques earned one point each and the Hashing techniques and the Count technique with Cosine Similarity earned 0 points, the Count technique with Sigmoid Kernel earned 9 points. Due to the counterintuitive

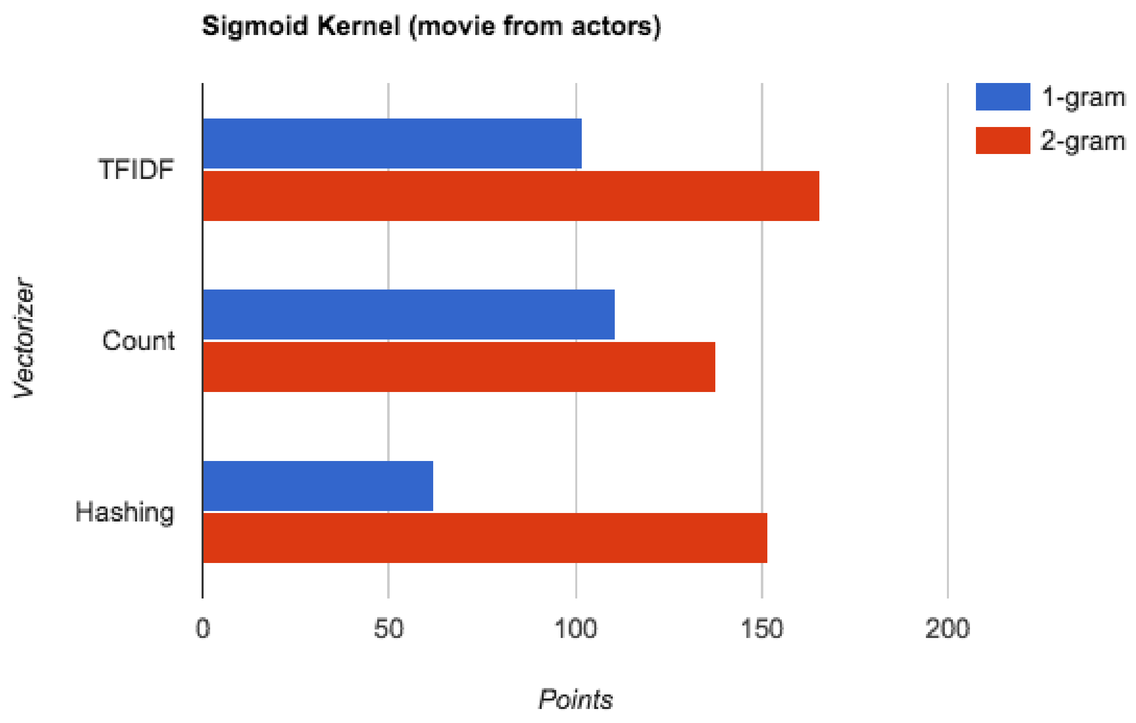


Figure 3: Results which we didn't expect

nature of these results, the code was double checked (the code is the same across all techniques with a few words changed) and the tests tried again several time; the results were validated.

Furthermore, when passed the query for "Casino Royale," the system was able to come up with relevant results, such as "Skyfall," "Spectre," and "The Golden Compass"; each of these guesses is correct for exactly one of the three actors mentioned in the query. On the other hand, when passed the query for "Inception," most of the guesses featured only one of the three actors.

Our suspicion is that the weakness in the methods in these cases is due to the fact that, given a query like "dicaprio winslet," the system believes (except in the TFIDF cases) that "dicaprio dicaprio," "dicaprio winslet," and "winslet winslet" are of equal relevance to the query (this generalizes easily to n-grams). Therefore, articles starring one of the actors, but mentioning him/her very many times, will be considered over the target article, which, although being very unique in that it mentions *all* actors, may only mention them a few times.

4 Future Work

There are a number of areas for future development. One problem with our current model is that 2-gram analysis on a query containing one word entities will perform undesirably. For example, the query "prince of egypt hugo searching for bobby fischer" will not be helped by the presence of "hugo", since it will stupidly look for "egypt hugo" and "hugo searching". A potential solution to this would be to vectorize with respect to 1-grams *and* 2-grams, but then we might run into problems with Hugo Weaving". This conundrum deserves more thought.

Currently, the program computes the matrices of vectors again for each query (since the first vector of the matrix represents the query, the matrix changes marginally for each query). It would speed up testing a great deal to have the matrices computed once, before all of the queries, and then updated for each query. Furthermore, higher level n-grams would be reasonable to implement.

Although this sounds simple, there are a few reasons it's a nontrivial task.

1. The vector representing the query needs to be the same shape as the matrix. For example, if there are 1000 tokens in the corpus, the size of the vector for the query needs be 1x1000.

2. If the query contains a token that doesn't appear in the corpus, the matrix needs to have a row added (or the word should be disregarded).

Furthermore, although we specifically concentrated our examples in cinema, the only detail of our corpus which we used to our advantage is that our desired system output is the title of one of the documents. Therefore, the problem could be generalized to any such corpus. An example would be identifying artists by their works, researchers by their discoveries, and more.

Finally, our 2-gram analysis was flawed in that, as a result of using the default skearn, we would split a query like "marion cotillard tom hardy" into "marion cotillard", "cotillard tom", "tom hardy". While the presence of "cotillard tom" affects neither our runtimes nor our results in our experiments, we would run into problems with a query like "elton john wayne newton". A potential solution to this could include the user inputting the elements of the query separately.

```
Using Count with sigmoid kernel
10 Skyfall
9  The_Hunt_(2012_film)
8  Sin_City:_A_Dame_to_Kill_For
7  300:_Rise_of_an_Empire
6  Cowboys_&_Aliens
5  The_Golden_Compass_(film)
4  Kingdom_of_Heaven_(film)
3  Spectre_(2015_film)
2  The_Salvation_(film)
1  Defiance_(2008_film)
```

Figure 4: Failure to find "Casino Royale"

Acknowledgments

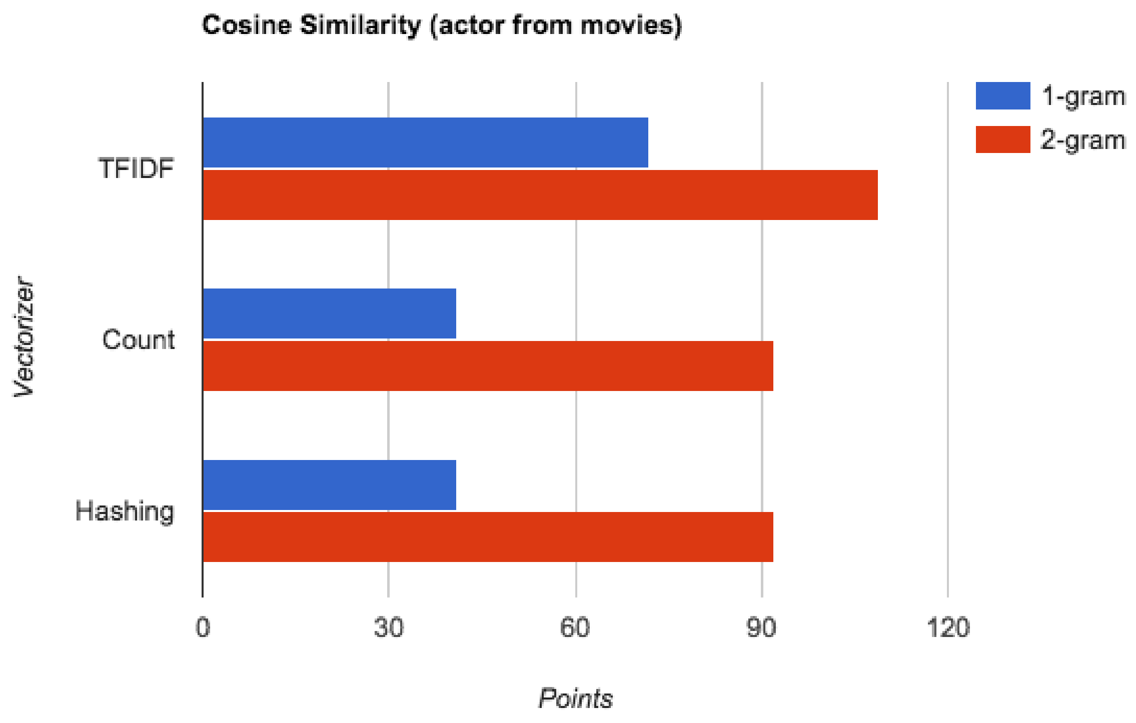
The author would like to thank Adam Meyers for his help and support and the survey subjects for their creative contributions.

References

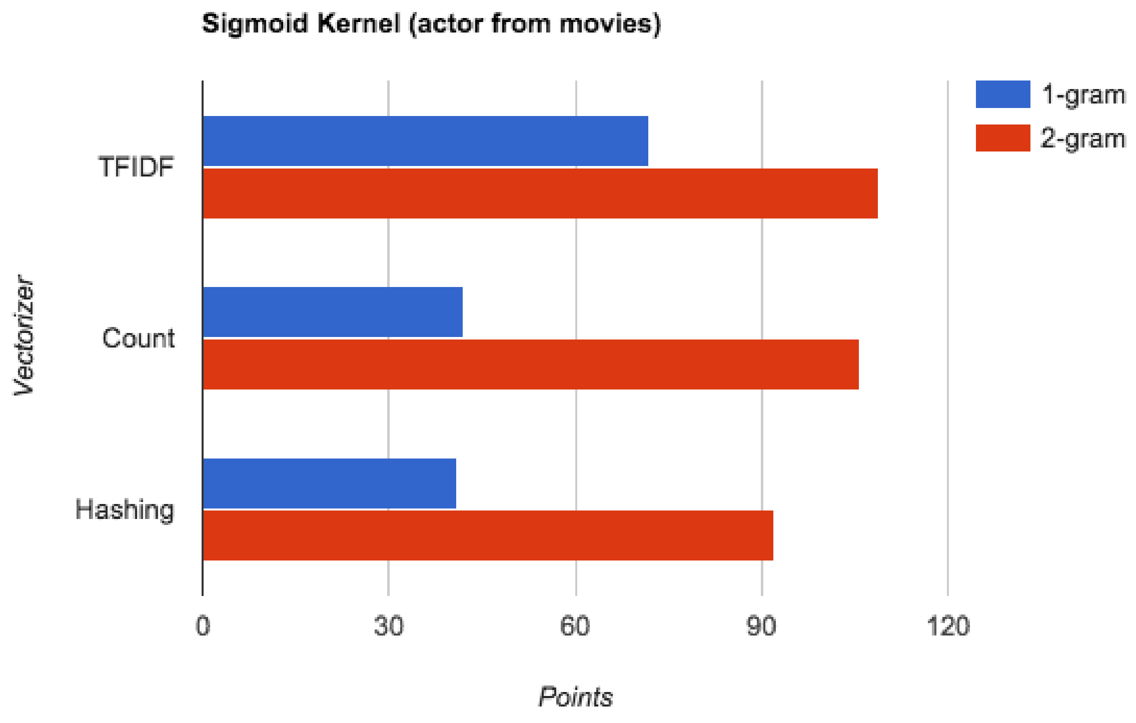
Jurafsky, Daniel, and James H. Martin. *Speech and Language Processing : An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Upper Saddle River, NJ: Prentice Hall, 2000. Print.

Bird, Steven, Ewan Klein, and Edward Loper. *Natural language processing with Python.* Beijing Cambridge Mass: O'Reilly, 2009. Print.

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.



Cosine Similarity (actor from movies)
1-gram
2-gram
TFIDF
Count
Hashing
Vectorizer
0
30
60
90
120
Points



Sigmoid Kernel (actor from movies)
1-gram
2-gram
TFIDF
Count
Hashing
Vectorizer
0
30
60
90
120
Points