Movie Information Web Search and Classification

Frankie Wu
fwu{at}cs{dot}stanford{dot}edu
Department of Computer Science
Stanford University
Palo Alto, CA USA

ABSTRACT

While there exist several web sites with movie information (cast and crew, etc.), none of these may be use in commercial applications. The same information is dispersed across the world wide web, but in much less predictable formats. This paper presents an investigation of using an existing web search engine to collect HTML pages based on movie titles, and extracting a small subset of the movie information. Classification is performed by machine learning though a maximum entropy Markov model (MEMM) and sequence labeling through a Viterbi decoder.

INTRODUCTION

Categorical information is available across the world wide web and could be used for commercial purposes if it could be collected and classified. One such example is movie information, such as cast and crew, film genre, country of origin, running time. While there are already several web sites which contain this information (IMDB, AllMovie, Yahoo! Movies, Amazon.com's DVD catalog), they may not be used for commercial purposes (other than their own). Also, the information is not complete across these databases, sometime in conflict, and their formats, though internally consistent, varies from website to website. In most cases, the information is not programmatically accessible.

An alternative to using any one of these websites, or even restricting oneself to dedicated movie websites, is to use the web as a whole. The challenge lies non-standardized format in which the data is presented. Here, the machine learning technologies of the maximum entropy Markov model (MEMM) and the Viterbi algorithm may provide a method for converting the raw, free form data into categorical lists of information.

The system presented here attempts to classify data for 20 movies, based on up to 50 HTML files (per movie) containing those movie tiles, after being trained on up to 80 movies using data from up to 50 HTML files per movie. The categories into which the data are classified are:

```
ACTOR
DIRECTOR
SCREENWRITER
PRODUCER
COMPOSER
```

Let us refer to this group collectively as the movie contributors.

The system consists of two stages: the first collects and annotates the data; the second performs the classification.

COLLECTION AND ANNOTATION

For this project, data collection is done through an existing web search engine: the Yahoo! Search API. In a production system, this would be replaced by an internally built web crawler. To reduce the emphasis on this part of the system and to concentrate on the machine learning aspects, the existing search engine is used. The process is automated by a Java program which takes as input movie titles and their years, and uses an HTTP URL to request a web search from the Yahoo! Search server. An example of a search request (for the movie Dinosaur Planet, 2003) is:

```
http://search.yahooapis.com/WebSearchService/V1/webSearch?appid=test&query="D
inosaur+Planet"+movie+2003&results=100&format=html
```

which returns data, in XML format, listing the URLs of the query results. These data are saved locally as XML files.

The Java program next parses these files, connects to their query result URLs, and downloads the referenced HTML files locally. Initially, 200 movies with 100 HTML files per movie were desired, however disk space limitations required that only files for 100 movies with 50 HTML files per movie could be downloaded. It turns out that this is more than enough data, as will become apparent in discussion of the classification stage of the system.

As these HTML files are the training and test sets for subsequent classification, they require annotation. The volume of data (potentially 5,000 files, though the actual number is less than this because not all 100 movies have 50 query results) requires that the annotation be done in a

somewhat automated fashion. This is done through a master file which contains hand compiled lists of the movie data. One example (abbreviated) entry is:

```
36,1992,Lady Chatterley
ACTOR,Joely Richardson
ACTOR,Sean Bean
ACTOR,James Wilby
ACTOR,P.J. Davidson
ACTOR,Soo Drouet
COMPOSER,Jean-Claude Petit
DIRECTOR,Ken Russell
PRODUCER,Michael Haggiag
SCREENWRITER,Ken Russell
SCREENWRITER,D.H. Lawrence
```

Note that the categories are somewhat misnamed (see the `D.H. Lawrence` entry). A better name for `SCREENWRITER` is simply `WRITER`; also, a better name for `ACTOR` is `PLAYER`, as not all of the movies are "acted" (e.g. `Nicki Doane in "Ashtanga Yoga: Beginner's Practice with Nicki Doane"`). The master file is used as template with which any matching strings in the HTML files are annotated. This is admittedly an imperfect process, as the source of information for the hand compilation is manual web search, the most common resulting web sites being IMDB, AllMovie, and Netflix, which may not contain all the relevant information in each of the 50 HTML files per movie. It is however the best solution given the time and labor constraints of the project.

CLASSIFICATION

A Java program performs the classification through an MEMM classifier and Viterbi decoder. The starting point for this program is the source code resulting from CS224N programming assignment 3 (PA3). The MEMM is given hand built features, tailored toward the classification task. This is the feature set used in the classification of the data into one of the movie contributor categories:

```
Group 1:    word itself
            previous label

Group 2:    left bigram
            right bigram
            left trigram
            right trigram
            middle trigram
```

```
Group 3:    is word name-shaped
            is left word name-shaped
            is right word name-shaped
            are left bigram words name-shaped
            are right bigram words name-shaped
            are left trigram words name-shaped
            are right trigram words name-shaped
            are middle trigram words name-shaped

Group 4:    is [Dd]irect in left words

Group 4+:   is [Ss]tar in left words
            is [Aa]ct in left words
            is [Cc]ompose in left words
            is [Pp]roduce in left words
            is [Ww]rit in left words
            is [Ss]creenwrit in left words
```

Group 1 is the default set from the PA3 skeleton code. Group 2 is similar to the `word itself` feature from group 1 in that the feature is the string of words itself, e.g. the word and the word to its right in the case of the `right bigram`. Group 3 features are binary feature indicating whether the word or group of words is name-shaped, that is, starts with the regular expression [A-Z][a-z]. The intuition behind group 3 is that movie contributor names are names. Groups 4 and 4+ features are binary features indicating whether some key words precede the given word in the sentence, that is, if "starring" appears before a word, that word is more likely to be an actor's name than if "starring" does not appear. This feature is very expensive computationally, and for those reasons, the results of testing only the single feature in Group 4 is presented. (The testing of all features in groups 4 and 4+ was attempted, but proved to be impractical.) Furthermore, for the same reason of computational expense, a maximum of 3 words to the left of the current word is searched for the key word of interest, rather than the entire string to the left as was originally intended.

Because the features are created effectively by (educated) guess work, the actual relative weight of each feature is optimized by maximizing the entropy of the Markov model. This is achieved by setting the derivative of the objective function to 0, where the objective function is[1]:

$$F(\vec{\lambda}) = [-1[\sum_{\langle c,d\rangle \in \langle C,D\rangle} \ln p(c \mid d, \vec{\lambda})]] + \sum_i \frac{\lambda^2{}_i}{2\sigma^2}$$

where the conditional probability p is given by:

$$p(c \mid d, \vec{\lambda}) = \frac{\exp[\sum_i \lambda_i f_i(c,d)]}{\sum_{c'} \exp[\sum_i \lambda_i f_i(c',d)]}$$

and the final term in the objective function:

$$\sum_i \frac{\lambda_i^2}{2\sigma^2}$$

is the smoothing term, in this case a Gaussian prior.

The derivative of the objective function with respect to the feature weight vector is then:

$$\frac{dF(\vec{\lambda})}{d\lambda_i} = -1[[\sum_{\langle c,d \rangle \in \langle C,D \rangle} f_i(c,d)] - [\sum_{\langle c,d \rangle \in \langle C,D \rangle} \sum_{c'} p(c' \mid d, \vec{\lambda}) f_i(c',d)]] + \frac{\lambda_i}{\sigma^2}$$

And setting this to 0 gives the iterative step in the maximum entropy Markov model classification:

$$0 = -1[[\sum_{\langle c,d \rangle \in \langle C,D \rangle} f_i(c,d)] - [\sum_{\langle c,d \rangle \in \langle C,D \rangle} \sum_{c'} p(c' \mid d, \vec{\lambda}) f_i(c',d)]] + \frac{\lambda_i}{\sigma^2}$$

$$\sum_{\langle c,d \rangle \in \langle C,D \rangle} f_i(c,d) = [\sum_{\langle c,d \rangle \in \langle C,D \rangle} \sum_{c'} p(c' \mid d, \vec{\lambda}) f_i(c',d)] + \frac{\lambda_i}{\sigma^2}$$

that is, to make the empirical count for each feature match the counts predicted by the model with the current feature weights (plus smoothing).

Once the feature weights are known, the Viterbi algorithm is used to sequence a test string into its component labels. The Viterbi algorithm is linear in the number of words in the test string and operates on the Markov property which allows the calculation of the most likely path to any given state in a sequence to be calculated as the most likely combination of 1) the path to any immediately prior state and 2) the transition from that prior state to the given state[2].

EXPERIMENTAL RESULTS

Initial results are from a test run using a training set of 80 movies with 50 files per movie and with only the default set of features, group 1 in the Classification section above. The test set consists of 20 movies with 50 files per movie.

80 movies, 50 files per movie	Precision		Recall		F-Measure
ACTOR	0.2776	(617/2223)	0.0953	(617/6476)	0.1419
COMPOSER	0.0000	(0/0)	0.0000	(0/832)	0.0000
DIRECTOR	0.0537	(62/1154)	0.0931	(62/666)	0.0681
PRODUCER	0.0000	(0/4)	0.0000	(0/112)	0.0000
SCREENWRITER	0.0000	(0/1)	0.0000	(0/493)	0.0000
TOTAL	0.2008	(679/3382)	0.0791	(679/8579)	0.1135

The results are somewhat disappointing, but perhaps to be expected due to the lack of hand designed features. The initial test also requires too much computation time, about an hour, to be a feasible test basis for experimenting with the different additional features. To reduce testing time, the training set is reduced. The test set is not changed for 2 reasons: first to provide a consistent test set so all results are comparable, and second because the size of the test set does not greatly affect computation time.

There is a choice between reducing the number of movies and reducing the number of files per movie. 2 experimental runs, one with 80 movies, 10 files per movie, the other with 10 movies, 50 files per movie, execute in the expected relative times, 8m 6s and 5m 1s. However the accuracy data show that more is not necessarily better:

10 movies, 50 files per movie	Precision		Recall		F-Measure
ACTOR	0.5000	(54/108)	0.0083	(54/6476)	0.0164
COMPOSER	0.0000	(0/0)	0.0000	(0/832)	0.0000
DIRECTOR	0.0000	(0/0)	0.0000	(0/666)	0.0000
PRODUCER	0.0000	(0/0)	0.0000	(0/112)	0.0000
SCREENWRITER	0.0000	(0/0)	0.0000	(0/493)	0.0000
TOTAL	0.5000	(54/108)	0.0063	(54/8579)	0.0124

80 movies, 10 files per movie	Precision		Recall		F-Measure
ACTOR	0.0639	(20/313)	0.0031	(20/6476)	0.0059
COMPOSER	0.0000	(0/0)	0.0000	(0/832)	0.0000
DIRECTOR	0.0000	(0/0)	0.0000	(0/666)	0.0000
PRODUCER	0.0000	(0/0)	0.0000	(0/112)	0.0000
SCREENWRITER	0.0000	(0/0)	0.0000	(0/493)	0.0000
TOTAL	0.0639	(20/313)	0.0023	(20/8579)	0.0045

The results of this side experiment may be interpreted as follows: The “depth” of any given movie file, that is, its number in the query result list, is a function of its query rank. The test set always goes 50 results deep, and so for each movie has a varied mix of quality with respect to the movie title query. The training set should match the range of these qualities at the expense of having more examples of different movies, all with considerably higher quality, even at the expense of fewer bytes of training data.

Returning to the original motivation behind this side experiment, let the remainder of the test runs use the more effective training set of 10 movies, 50 files per movie. Due to the sparseness of labels the training data, especially considering the reduced training set, all the remaining tests will show no little or no activity for any category other than `ACTOR`. (Surprisingly, even with the addition of the `"Is [Dd]irect in left words"` feature in group 4 above, there was no activity in the `DIRECTOR` category; this can perhaps be attributed to the sparseness of data in the reduced training set as well.) Therefore, only the results of the `ACTOR` category are discussed.

Borrowing an approach from a paper by Finkel and Manning[3], the effect of each individual feature is examined by turning only that feature off. The following table represents the results of test runs incorporating every feature in groups 1-4 above except the one indicated:

ACTOR Category Only					
Feature Removed	**Precision**		**Recall**		**F-Measure**
none (all features present)	**0.6434**	**(166/258)**	**0.0256**	**(166/6476)**	**0.0493**
word	0.7253	(66/91)	0.0102	(66/6476)	0.0201
previous label	0.5693	(152/267)	0.0235	(152/6476)	0.0451
left bigram	0.4793	(81/169)	0.0125	(81/6476)	0.0244
right bigram	0.6327	(186/294)	0.0287	(186/6476)	0.0549
left trigram	0.5762	(121/210)	0.0187	(121/6476)	0.0362
right trigram	0.5495	(100/182)	0.0154	(100/6476)	0.0300
middle trigram	0.5693	(78/137)	0.0120	(78/6476)	0.0236
word name shaped	0.6364	(161/253)	0.0249	(161/6476)	0.0479
left word name shaped	0.5482	(125/228)	0.0193	(125/6476)	0.0373
right word name shaped	0.5824	(152/261)	0.0235	(152/6476)	0.0451
left bigram name shaped	0.5949	(188/316)	0.0290	(188/6476)	0.0554
right bigram name shaped	0.5445	(153/281)	0.0236	(153/6476)	0.0453
left trigram name shaped	0.6197	(132/213)	0.0204	(132/6476)	0.0395
right trigram name shaped	0.6085	(143/235)	0.0221	(143/6476)	0.0426
middle trigram name shaped	0.6190	(143/231)	0.0221	(143/6476)	0.0426
[Dd]irect in left words	0.6007	(161/268)	0.0249	(161/6476)	0.0477

The data show that the most effective feature overall is the word itself. Note however that while it has a great affect on recall, it actually has a detrimental effect on precision. A reasonable explanation is that the same name appearing has a higher than average likelihood of referring to the same actor, hence elimination of that feature would diminish recall. However, the same name might refer to a non-actor as well, hence the detrimental affect on precision.

CONCLUSIONS AND FUTURE WORK

The next logical step is to take the feature set optimized by the experiments with the reduced training set and run a test using the full training set of 80 movies and 50 files per movie. The results from the initial (and only) test using the full training set and the 2 default features are 3 times better than the best test results using the reduced training set and hand crafted features (which in turn are 3 times better than the results from reduced training set and 2 default features). Actually, this was attempted without success and put on hold due to difficulties with the test server. Still, while the absolute scores are disappointing, evidence of improvement through feature design and through scaling of the training set is encouraging.

It is also worthwhile to revisit the original motivation behind this project. The success metric discussed in this paper is based on matching every instance of a labeled word. Since the ultimate goal is to obtain a list of cast and crew members for a given movie, perhaps a better metric is to have the system output such a list and compare that against the actual list of cast and crew members. This would change the approach of feature design because in this scenario, it doesn't matter if a few instances of an actor are missed as long as at least one is found, that is, the recall scores, as discussed above, are not so important, but the importance of precision is emphasized.

Finally, I would be remiss if I did not confess to the (ultimately resistible) urge to name this system Star-TREC.

AKNOWLEDGEMENTS

Thank you to Professor Christopher D. Manning and Pi-Chuan Chang for guidance in this project. And thank you to Paul Baumstarck and Professor Manning again for their insightful lectures on MEMMs.

REFERENCES

[1] Christopher D. Manning, Paul Baumstarck, Pi-Chuan Chang: **CS 224N / Ling 280 Assignment 3: MaxEnt Sequence Classifiers & Treebank Parsing** 2008:3-4.

[2] Stuart Russell, Peter Norvig, **Artificial Intelligence, A Modern Approach, Second Edition** 2003: 547-549.

[3] Jenny Finkel, Shipra Dingare, Christopher D. Manning, Malvina Nissim, Beatrice Alex, Claire Grover, **Exploring the Boundaries: Gene and Protein Identification in Biomedical Text**.