Classifying Movie Scripts by Genre with a MEMM Using NLP-Based Features

Alex Blackstock Matt Spitz

6/4/08

Abstract

In this project, we hope to classify movie scripts into genres based on a variety of NLP-related features extracted from the scripts. We devised two evaluation metrics to analyze the performance of two separate classifiers, a Naive Bayes Classifier and a Maximum Entropy Markov Model Classifier.

Our two biggest challenges were the inconsistent format of the movie scripts and the multiway classification problem presented by the fact that each movie script is labeled with several genres. Despite these challenges, our classifier performed surprisingly well given our evaluation metrics. We have some doubts, though, about the reliability of these metrics, mainly because of the lack of a larger test corpus.

1 Introduction

Classifying bodies of text, by either NLP or non-NLP features, is nothing new. There are numerous examples of classifying books, websites, or even blog entries either by genre or by author [7, 8]. In fact, a previous CS224N final project was centered around classifying song lyrics by genre [6].

Despite the large body of genre classification in other types of text, there is very little involving movie script classification. A paper by Jehoshua Eliashberg [1] describes his work at The Wharton School in guessing how well a movie will perform in various countries. His research group developed a tool (MOVIEMOD) that uses a Markov chain model to predict whether a given movie would gross more or less than the median return on investment for the producers and distributors. His research centers on different types of consumers and how they will respond to a given movie.

Our project focuses on classifying movie scripts into genres purely on the basis of the script itself. We convert the movie scripts into an annotated-frame format, breaking down each piece of dialogue and stage direction into chunks. We then classify these scripts into genres by observing a number of features. These features include but are not limited to standard natural language processing techniques. We also observe components of the script itself, such as the ratio of speaking to non-speaking frames and the average length of each speaking part. The classifiers we explore are the Maximum Entropy Markov Model from Programming Assignment 3 and an open-source Naive Bayes Classifier.

2 Corpus and Data

The vast majority of our movie scripts were scraped from online databases like dailyscript.com, and other sites which provide a front-end to what is apparently a common collection of online hypertext scripts, ranging from classics like *Casablanca* (1942) to current pre-release drafts like *Indiana Jones 4* (2008). Our raw pull yielded over 500 scripts in .html and .pdf format, the latter of which had to be coerced into a plain text format to become useful. Thanks to surprisingly consistent formatting conventions, that vast majority of these scripts were immediately ready for parsing into object files. However, some of the scripts varied in year produced, genre, format, and writing style. The latter two posed significant problems for our ability to parse the scripts reliably. After discarding the absolutely incorrigible data, we were left with 399 scripts to be divided between train and test sets.

The second piece of raw data we acquired was a long-form text database of movie titles linked to their various genres, as reported by imdb.com.

The movies in our corpus had 22 different genre labels. The most labels any movie had was 7, the fewest was 1, and the average was 3.02. The exact breakdown is given below:

2.1 Processing

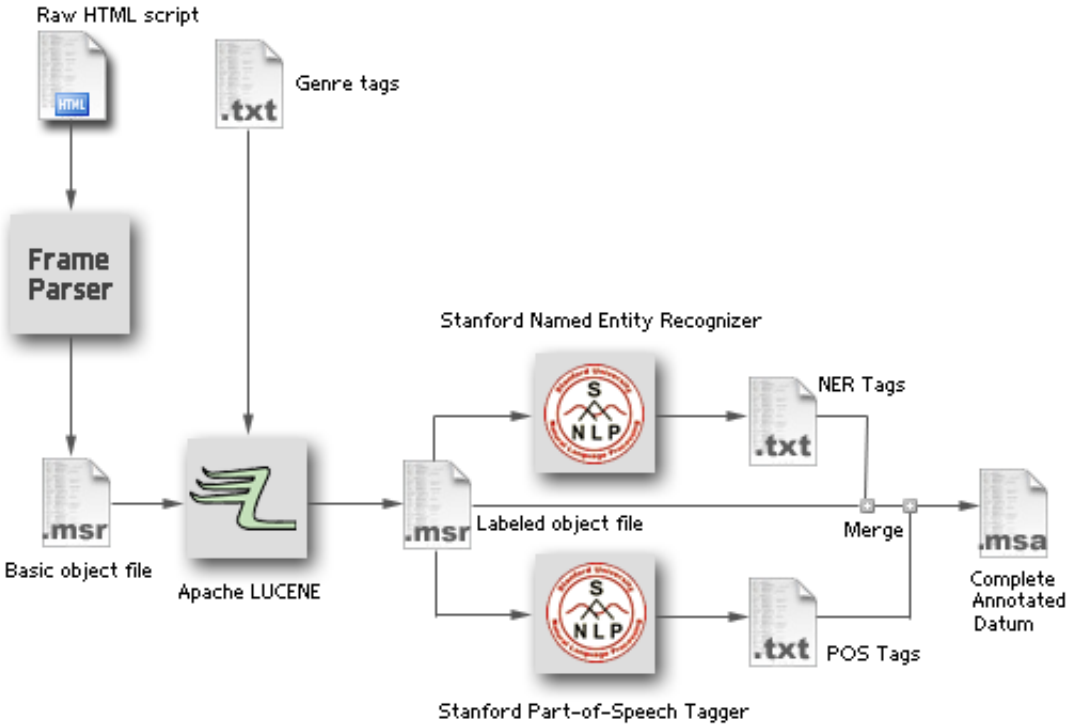
The transformation of a movie script from a raw text file to the complex annotated binary file we used as datum during training required several rounds of pulling out higher-level information from the current datum, and adding that information back into the script. Our goal was to compute somewhat of an "information closure" for each script to maximize our options for designing helpful features.

Genre	Count
Drama	236
Thriller	172
Comedy	119
Action	103
Crime	102
Romance	76
Sci-Fi	72
Horror	71
Adventure	62
Fantasy	48
Mystery	45

Genre	Count
War	17
Biography	14
Animation	13
Music	12
Short	11
Family	10
History	6
Western	5
Sport	5
Musical	4
Film-Noir	3

Table 1: All genres in our corpus with appearance counts

Building a movie script datum



The first step was to use various formatting conventions in the usable scripts to break each movie apart into a sequence of "frames," consisting of either character dialogue tagged with the speaker, or stage direction / non-spoken cues tagged with the setting. The generated list of frames, which still consisted only of raw text at this point, were serialized into a .msr binary object file.

Raw frames:

```
FRAME--------------------------
        TYPE: T_OTHER
        Setting: ANGLE ON:  A RING WITH DR. EVIL'S INSIGNIA ON IT.
        Text:  THE RINGED HAND IS STROKING A WHITE FLUFFY CAT
-------------------------------

FRAME--------------------------
        TYPE: T_DIALOGUE
        Speaker: DR. EVIL
        Text: I spared your lives because I need you to help me rid the world of
          the only man who can stop me now. We must go to London.
          I've set a trap for Austin Powers!
-------------------------------
```

```
... ... ...

FRAME--------------------------
        TYPE: T_DIALOGUE
        Speaker: SUPERMODEL 1
        Text:  Austin, you've really outdone yourself this time.
-------------------------------

FRAME--------------------------
        TYPE: T_DIALOGUE
        Speaker: AUSTIN
        Text:  Thanks, baby.
-------------------------------
```

Using the textual search capabilities of Lucene (discussed below), we then paired the .msr files with the correct genre labels, to be used in training and testing. Finally, the text content of each labeled .msr was run through the Stanford NER system [2] and the Stanford POS tagger [9], generating two output files with the relevant part-of-speech and named entity tags attached to each word. The .msr was annotated with this data and then re-serialized, producing our complete .msa ("movie script annotated") object file to be used as a datum.

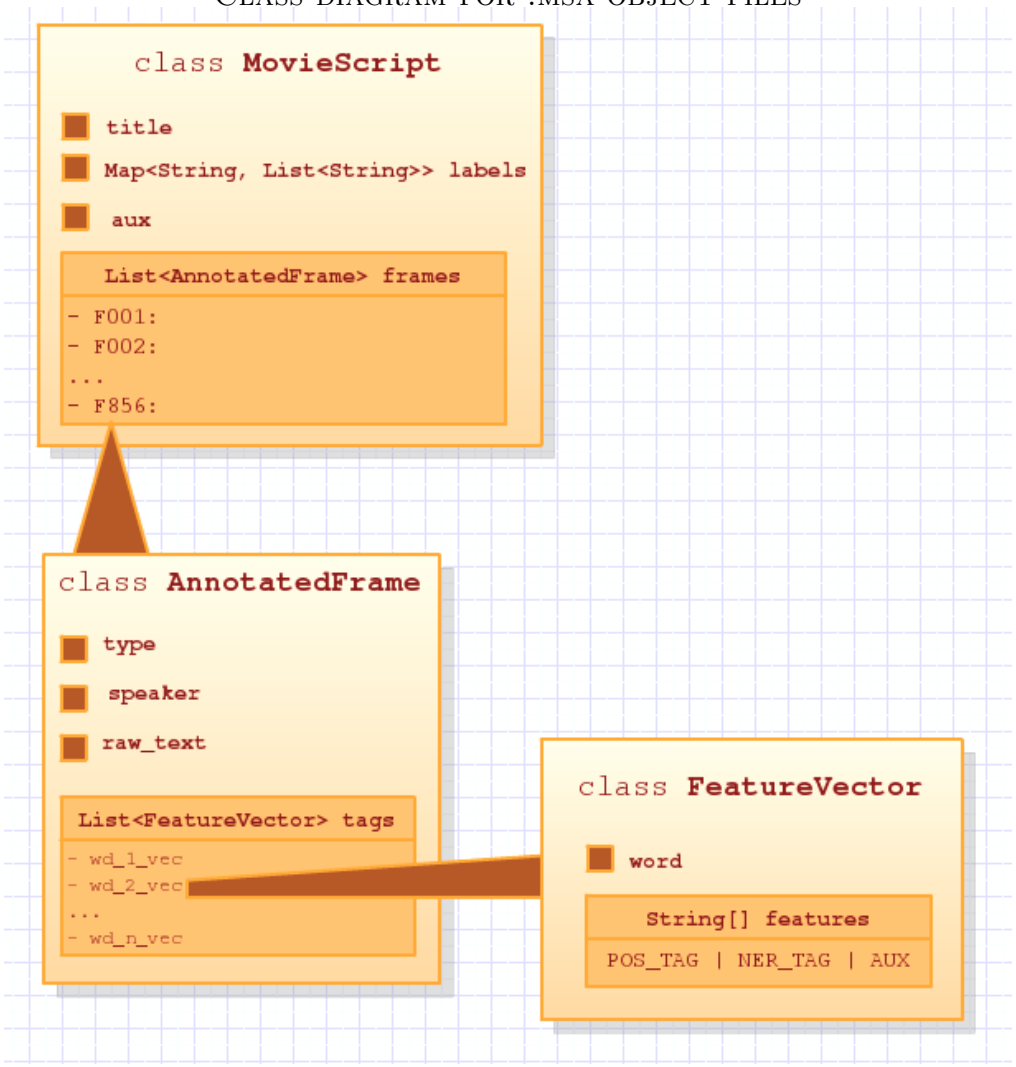
Annotated frames:

```
FRAME--------------------------
    TYPE: T_OTHER
    Setting: INT.  DR. EVIL'S PRIVATE QUARTERS
    Text: Dr. Evil, Scott and the evil associates finish dinner.

Dr.: [NNP][PERSON]
Evil,: [NNP][PERSON]
Scott: [NNP][PERSON]
and: [CC][O]
the: [DT][O]
evil: [JJ][O]
associates: [NNS][O]
finish: [VBP][O]
dinner.: [NN][O]
-------------------------------
... ... ...
FRAME--------------------------
        TYPE: T_DIALOGUE
        Speaker: BASIL EXPOSITON
        Text: Our next move is to infiltrate Virtucon. Any ideas?

Our: [PRP\$][O]
next: [JJ][O]
move: [NN][O]
is: [VBZ][O]
to: [TO][O]
infiltrate: [VB][O]
Virtucon.: [NNP][ORGANIZATION]
Any: [DT][O]
```

```
ideas?: [NNS][O]
--------------------------------
```

A particular challenge in this final step was aligning the output tags with our original raw text. The Stanford classifiers tokenize raw text by treating punctuation and special characters as taggable, while we were only interested in the semantic content of the actual space-delimited tokens in the dialogue and stage direction.

Class diagram for .msa object files



2.2 Lucene

As mentioned above, the IMDB genre database is stored in a relational format with genres for each entry in the IMDB. Given the inconsistent, sloppy format of the movie scripts and the existence of duplicate movie titles, we couldn't use exact-text matching to pull the genres. Instead, we enlisted the help of Apache Lucene, an open-source Java search engine.

To begin, we indexed the movie titles in Lucene [3]. Then, for each movie script, we searched for its title in the Lucene database. We go over the results by hand, consulting the original script, and pick the best result. Lucene built the bridge between the movie script data and the IMDB data. Fortunately, we kept the work we had to do by hand to a minimum while maintaining high accuracy in our labels.

2.3 Dividing the data

Once we had properly-formatted and tagged movie scripts, we divided them into test and training sets. Our two main goals here were randomness and repeatability. Thus, we have the user specify a seed with each run to determine how to divide the test and training sets. The divisions are random given a seed, but when the same seed is provided, the test and training sets produced are identical. This offered the randomness and repeatability that we were looking for.

Of the 399 movie scripts, we chose a test set percentage of 10%, giving us 40 movie scripts in the test set and 359 movie scripts in the training set. We felt that this was an acceptable percentage, especially considering that we didn't have very much data.

3 Features

Our guiding intuition in feature design was to pick out orthographic and semantic features in the text, as well as statistics about these features, which tightly correlate with a specific genre or class of genres. Since we have only scripts as training data about the movies (no cinematographic or musical cues), our classifier had to perform the same high level analysis that one does when reading a book and attempting to form mental representations of the characters and plot. Is there a lot of descriptive language (high JJ ratio)? Is the plot dialogue driven, or scene driven? Are the conversations between characters lively and fast paced, or more like monologues? Is there frequent mention of bureaucracy or other organizations? Does the language of individual characters identify their personality or guide the plot? All of these

questions help human readers mentally represent a story, and thus we investigated how implementing these observations as features effected our classifier's ability to do the same.

3.1 High-Level NLP

Most of our features were computed, continuous valued ratios, and thus the strategy we often applied (to avoid an overwhelming sparsity of feature activation) was to bucketize these numbers into categories representing an atypically low value, an average value, a high value, and an atypically high value. The statistics to determine where the bucket boundaries fell were obtained by averaging over our whole data set. We present here the features that utilized this method for our NER and POS annotations.

- F_[LOW,AVERAGE,HIGH]_JJNN_RATIO
  Taking the script as a whole, what is the ratio of descriptive words to nominals? Here, JJ refers to the sum off JJ,JJR, and JJS counts, while NN includes all forms of singular and plural nouns. This feature was designed to partition the scripts into a batch with laboriously spoken lines and lots of detailed scene description (drama? romance? fantasy?) and ones with minimal verbosity (action? thriller? crime? western?).

- F_[LOW,AVERAGE,HIGH]_PERPRON_RATIO
  Of all the spoken words, what proportion were personal pronouns like *I*, *you*, *we*, *us*? Indicative of a romance or drama.

- F_[LOW,AVERAGE,HIGH]_W_RATIO
  Of all the spoken words, what proportion were wh-determiners, wh-pronouns, and wh-adverbs? All of our favorite crime, mystery, and horror films have phrases like, *which why did he go?*, or *who can we trust?*, or *where is the killer?*

- F_[LOW,AVERAGE,HIGH,VHIGH]_LOC
  On the script as a whole, is there a preponderance of location names, like *Iceland*, or *San Francisco*? In particular, can we detect dated place names like *USSR*, or *Stalingrad* to pick out history, war, or biography films?

- F_[LOW,AVERAGE,HIGH,VHIGH]_ORG
  Does the script and its characters make mention of many organizations, particularly governmental, like the *FBI*, *KGB*, or *NORAD*? These sorts of acromyns

are labeled with high accuracy by the Stanford NER system, and correlate well with action/espionage type films.

- F_[LOW,AVERAGE,HIGH,VHIGH]_PER
  Perhaps the most trivial of the NER-based features, but included for completeness. Has the ability to detect whether a particular script is person/character rich (a musical for instance), or if it is more individual centered (thriller, e.g.).

- F_[LOW,AVERAGE,HIGH]_EXCLAIM_PCT
  This feature aggregates all words and phrases in the script that could be construed as exclamations of some sort (such as combinations of UH and RB/RBR tagged words, and imperative form verbs). Exclamations carry with them the feel of comic-book style action and include the onomatopoeia words often used in stage cues, such as *CRASH*, *Oh, no!*, and *Hurry!*.

3.2 Character Based NLP

These features are computed by first identifying the major/important agents in the movie via percentage of speaking parts, pulling out all of their frames, and attempting to draw a characterization of their personality based on the kind of language they use in these frames. For example, if we have two main characters in a film, one of whom speaks very curtly and one who rambles on and on, a whole-script analysis might only reveal that an average amount of descriptive language was used. We gain much more information, however, if we can say that this is because we have one happy-go-lucky motor mouth and, perhaps, their dual: a stoic, introvert. Drastically polarized characters are often used for dramatic effect (e.g. romance) and for comic relief (comedy). Each of these features says: "this movie has a main character who ..."

- F_[LOW,HIGH]_ADJ_CHARACTER
  Identifies a character who uses more, or less, adjectives than average (i.e. a monologue-er).

- F_[LOW,HIGH]_PERPRON_CHARACTER
  A hopeless romantic, bent on the use of personal and reflexive pronouns: "We were made for each other."

- F_[LOW,HIGH]_ADV_CHARACTER
  Adverb usage analyzer: "come quickly!"

- F_[LOW,HIGH]_EXCLAIM_CHARACTER
  Used as a per-character version of the global exclamation identifier described above. Do we have a character that is just too excited for their own good?

- F_[LOW,HIGH]_VPASTPART_CHARACTER
  Use of the past participle conjugation often indicates a more refined (or alternatively, archaic) manner of speaking: "have you eaten this morning?"

- F_[LOW,HIGH]_MODAL_CHARACTER
  Characteristic of sage-like advice: "of course you can, but should you?"

- F_[LOW,HIGH]_ORG_CHARACTER
  Per-character implementation of the global NER-organization feature. Do we have an informant, mole, or law enforcer? "Agent Dean of the NSA. The State Department sent us."

- F_[LOW,HIGH]_PER_CHARACTER
  Per-character implementation of the global NER-person feature.

- F_[LOW,HIGH]_LOC_CHARACTER
  Per-character implementation of the global NER-location feature. "Toto, I don't think we're in Kansas anymore"

3.3 Non-NLP

These features do not make use of the NER and POS information generated by the Stanford taggers. Instead, they survey high-level information about the composition of the frames in an attempt to pick out the idiosyncrasies of various genres.

- F_[LOW,AVERAGE,HIGH,VHIGH]_FRAMETYPE_RATIO
  This feature is a bucketized representation of the ratio of dialogue frames to non-dialogue frames. That is, do we have a film with many complex/interwoven scene transitions, or are we dealing with long scenes with lots of back-and-forth conversation?

- F_[LOW,AVERAGE,HIGH,VHIGH]_AVDLEN_RATIO
  Bucketized representation of the average dialogue length, computed by summing the number of tokens in all frames of type T_DIALOGUE and dividing by the number of such frames. This feature is good at picking out films with lots of sequences of quick and pithy conversations.

- F_[LOW,AVERAGE,HIGH,VHIGH]_NUM_SPK
  Bucketized number of characters with major speaking roles. Are we in a documentary setting with many speakers, changing all the time, or is this a chronicle of the romance between two leading characters?

- F_NUM_MAIN_CHARACTERS=n
  The litteral number of main characters. Since in practice our algorithm for finding this number never returned more than four or five, we were sure that this feature wasn't contributing to sparsity of feature activation.

- F_[LOW,AVERAGE,HIGH]_MAINS
  A bucketized version of the litteral number above.

4 MEMM Classifier

The Maximum Entropy Markov Model is based on the MaxEnt model, in which we use a logistic regression to assign weights to features. The difference is that the MEMM also includes the Viterbi algorithm, which allows us to condition the way in which we label data based on previous labels. Rather than re-invent the wheel, we chose to start with our existing code from the Programming Assignment 3 MEMM classifier.

One of the less-than-ideal consequences of using a MEMM backed by the Viterbi algorithm to classify our scripts is that we lose the ability to condition on a series of previous classification decisions. This functionality obviously remains in tact from our previous implementation, but in the context of making n-ary decisions on one movie script at a time, the information is useless. Since each decision is independent of the next, we know that the probability of labeling a script with class c, $P(c)$, is equal to the probability of labeling the script with c given that the previous decision was d, $P(c \mid d)$. Since $P(c) = P(c \mid d)$, we discard d. An investigation and discussion of the empirical consequences of this design decision takes place later on. For now, we discard the Viterbi component and instead just take the first guess. Thus, we're essentially stripping our MEMM back down to a MaxEnt. This is a little sloppy, but it allowed us to re-use our existing code and instead focus on coming up with interesting features.

4.1 Procedure

Our "MEMM" (really, it's just MaxEnt) uses logistic regression to learn weights for the selected features. Once it has those weights, it uses getLogProbabilities to apply

the weights to the test datum's features and assigns the probabilities that the datum is of each label type. Our MEMM ignores the Viterbi step and instead just takes the first round of guesses.

In the end, our classifier generates a set of sorted scores. We modified the original MEMM code (which only returned a single a label) and instead took the k-best scores to be our guessed labels, where k depended on our evaluation metric. We then compared these against the gold labels for our script to evaluate our classifier. We show an example of this classification below.

```
Blade II
[Action, Thriller, Horror]
Guess: [Action, Adventure, Horror, Thriller, Drama, | Sci-Fi, Crime, Short, Romance, Comedy]

Pulp Fiction
[Crime, Drama]
Guess: [Drama, Comedy, Romance, | Crime, Thriller, War, Mystery, Action, Music, Sci-Fi]

Sphere
[Horror, Sci-Fi, Thriller]
Guess: [Fantasy, Thriller, Drama, Horror, Action, | Comedy, Romance, Crime, Sci-Fi, Mystery]
```

The above example shows the movie, the gold labels (as chosen by IMDB) and then the top ten guesses as to which genre it could be. The pipe denotes the maximum number of allowed guessed labels for the PC Score (see below).

4.2 Evaluation Metrics

We used two different evaluation metrics to determine the effectiveness of our features. One of these metrics is the Partial Credit (PC) Score, which is designed to be a relaxed version of the F1 Score. Note that there is no notion of ordering in the IMDB genre labels. That is, if a movie's gold labels are Sci-Fi and Horror, no preference is given to either of these in our evaluation metric, and the order is irrelevant.

4.2.1 F1 Score

The F1 Score, as used before in class, is calculated as

$$\frac{2PR}{P+R}$$

However, since we assign more than one genre label to each datum, we needed a unique way to determine precision and recall. So, for each assigned gold label in a given datum, we increment a counter for that genre ("gold"). Additionally, we increment a counter if this label also appears in the guessed labels (i.e. it is "correct")

and a counter that represents how many times we guessed this label ("guessed"). In the end, we have three counters that contain how many times a genre actually appeared, how many times we guessed it appeared, and how many times we guessed correctly. Thus, we end up with an F1 Score for each genre. We take the average of the genre precision scores (weighted by how many times the genre was guessed) and the genre recall scores (weighted by how many times the genre was a gold label) to determine the overall F1 score for the metric. Our algorithm is below:

```
foreach datum
        foreach goldlabel
                goldcounter(label)++
                if guessedlabels contains label
                        correctcounter(label)++

        for i=0; i<goldlabels.size; i++
                guessedcounter(guessedlabel[i])++

foreach genre
        calculate precision and recall
        totalPrecision += genrePrecision
                                * (guessedcounter.count(genre) / guessedcounter.totalCount)

        totalRecall += genreRecall
                                * (goldcounter.count(genre) / goldcounter.totalCount)

overallF1 = (2*totalPrecision*totalRecall)/(totalPrecision + totalRecall)
```

We only consider the first n guessed labels, where n is how many gold labels have been applied. Thus, if a golden genre appears $(n+1)^{th}$ on the list of guessed labels, our F1 Score suffers as much as if that golden genre had appeared last in the guessed labels. To ease this penalty, we came up with the Partial Credit Score.

4.2.2 Partial Credit Score

The Partial Credit (PC) Score is based on the notion that guessed labels should have the flexibility to make some error without too harsh a penalty. After all, the classifier is asked to label a given movie script with up to six labels, some of which may not be closely related (Drama and Sci-Fi, for example). To calculate the PC Score, we take into account the order of the allowed guessed labels. The allowed guessed labels are the labels that we will consider against our gold labels. Specifically, we consider the first $\lceil g * 1.5 \rceil$ guessed labels, where g is the number of gold labels for this datum.

The intuition of the PC Score is that a guessed label is given a higher score if it appears closer to the front of the guessed labels list. So, we go over all gold labels and sum up the score for each label, where the score for a label is given by

$$\max(\#\text{ allowed guessed labels} - \text{position in guessed labels}, 0)$$

The maximum score for a given datum (if # allowed guessed labels is a) is

$$a + (a-1) + (a-2) + (a-n)$$

where n is the number of gold labels.

The PC Score, then, is computed by dividing the given datum score by the maximum possible score for a given datum. Our algorithm is below:

```
totalPCScore = 0
foreach datum
        allowedLabels = ceiling(goldLabels.size*1.5)
        maxPossible = 0
        thisScore = 0
        for int i = 0; i < goldLabels.size(); i++
                maxPossible += (allowedLabels - i)
                if (guessedLabels.contains(goldLabels[i])
                        index = guessedLabels.indexof(goldLabels[i])
                        thisScore += max(0, allowedLabels - index)
        thisPCScore = thisScore / maxPossible
        totalPCScore += thisPCScore / #test datums
```

Unsurprisingly, the PC Score is always greater than or equal to the F1 Score, since it allows for more error.

4.3 Results

To experiment with our features, we chose various subsets. The four main components are:

- Non-NLP
- Basic-NLP
- Part of Speech
- Named Entity Recognition

POS and NER features are done across the entire script, but they can also be done for individual speaking parts. Unless otherwise stated, each test below also has POS and NER for each individual speaker.

Feature Set	**PC Score**	**F1 Score**
all	0.52588	0.47244
all-no-speaker	0.52588	0.47244
base-nlp-and-ner	0.52388	0.45669
base-nlp-and-ner-no-speaker	0.52943	0.46457
base-nlp-and-pos	0.53327	0.48031
base-nlp-and-pos-no-speaker	0.60133	0.55147
base-nlp-and-pos-and-ner	0.53777	0.46457
base-nlp-and-pos-and-ner-no-speaker	0.54388	0.47244
just-base-nlp	0.56310	0.51969
just-ner	0.50371	0.43307
just-ner-no-speaker	0.50371	0.43307
just-non-nlp	0.54960	0.48819
just-pos	0.4882	0.56498
non-nlp-and-base-nlp	0.55861	0.48031
non-nlp-and-ner	0.52922	0.47244
non-nlp-and-ner-no-speaker	0.52922	0.47244
non-nlp-and-pos-and-ner	0.53269	0.46457

Table 2: MEMM classifier results using various sets of features

4.4 Analysis

Overall, our MEMM classifier performed rather well, considering that there were 22 genres to choose from and the genres we chose were correct with about 50% accuracy. Two important points to discuss are the performance of individual feature sets and the classifier itself.

We were very pleased that the NLP features performed relatively well (and even better than the non-NLP features!). We were concerned that these features would at best have no effect on our results given the different writing styles and the wide inconsistencies among scripts. However, one thing that we found rather disconcerting is that all sets of features are very close to one another. We did try the "perfect" feature set (using the labels as features) and the "worst-case" feature set (no labels at all). They behaved as expected (the first with 100% accuracy and the second just guessing the most frequent genres). This suggests that none of the features are performing particularly well.

Given that the feature sets were not very well differentiated, we suspected that something was wrong with our classifier, even though the "perfect" and "worst-case" scenarios behaved as expected. We suspected that in applying the classifier to our data and discarding the context that the Viterbi algorithm uses, we had messed something up or were not using it correctly. So, we tried a new classifier on our data with the same features to analyze how well it performed on the dataset.

5 Naive Bayes Classifier

Our motivation for attempting an entirely separate classification method was not simply to improve our ability to correctly classify scripts. Indeed, we did not ever expect an NBC to perform strictly better than our MEMM, or we would have pursued this classification method first. Rather, we were curious if we could verify our earlier intuition: that we were depriving the MEMM of critical inference capabilities by removing its ability to condition on a sequence of previous classification decisions (as in POS tagging applications). To take a discriminative model like our MEMM and reduce the amount of information in the data d of $P(c \mid d)$ by ignoring previous decisions, we thought, would result in performance similar to that of a generative model such as the tried-and-true naive Bayes classifier, which doesn't condition on the data at all when computing $P(c, d)$.

Our hope was that, were we to get similar precision, accuracy, and F1 scores from the generative model, then we could conclude with more certainty that at least some portion of our MEMM's performance flaws can be attributed to the removal of

the "prevLabel" conditioning feature that was so instrumental to the success of our MEMM when it was applied to biological NER tasks in the past.

5.1 jBNC

The generative classifier we wrote extended code from the open source jBNC Toolkit [5]: a java implementation of several naive Bayes classifiers which makes up a portion of the University of Waikato's WEKA machine learning project. We selected this implementation based on it's proven rubustness, particularly through its use in CMU's JavaBayes Project.

5.1.1 C4.5 Format

The jBNC classes utilize the C4.5 machine learning data format [4] to perform classification. Part of our implementation involved conjuring the contents of our .msa object files to conform to this standard.

```
EXAMPLE .names FILE:

Action, Adventure, Sci-Fi, Musical, Mystery, Drama, Western, Romance, Fantasy, Comedy, Horror, ...
adj_to_noun: low,high,avg.
ner_org: low,high,vhigh,avg.
frame_ratio: low,high,vhigh,avg.
av_dialog_len: low,high,vhigh,avg.
num_mains: low,high,avg.
exclaim_pct: low,high,avg.
high_adj_char: yes,no.
...
```

To generate C4.5 output files for each set of candidate features, a .names file was hand-coded, and we used our MEMM classifier to report which of the relevant features each movie script activated. Thus, each script was boiled down to one line of activated features in either the .data or .test file:

```
EXAMPLE .test FILE:

avg,avg,avg,avg,high,avg,high,avg,high,yes,yes,yes,no,yes,yes,yes,yes,yes,Crime
avg,avg,avg,avg,low,vhigh,high,avg,avg,yes,no,yes,yes,yes,yes,no,yes,yes,Action
avg,avg,avg,avg,low,low,high,low,high,yes,yes,yes,yes,yes,yes,yes,yes,yes,Biography
avg,avg,avg,vhigh,low,low,high,low,avg,yes,yes,yes,yes,yes,yes,no,yes,no,Action
avg,avg,avg,low,low,vhigh,high,low,low,yes,yes,yes,no,yes,yes,yes,no,no,Animation
avg,high,avg,high,low,avg,avg,avg,high,no,yes,yes,no,yes,yes,no,yes,yes,Drama
avg,avg,high,avg,low,avg,avg,high,avg,yes,yes,yes,yes,yes,yes,yes,no,yes,Romance
...
```

5.2 Procedure and Metrics

The jBNC Toolkit supports seven different implementations of the naive Bayes classifier, and five different performance evaluation metrics:

5.2.1 Classifiers

Naive Bayes : standard generative naive Bayes implementation

$$P(c \mid f_1, f_2, ..., f_n) = \frac{P(c)P(f_1, f_2, ..., f_n \mid c)}{P(f_1, f_2, ..., f_n)}$$

TAN : tree augmented naive Bayes

FAN : forest augmented naive Bayes

STAN : selective tree augmented naive Bayes

STAND : selective tree augmented naive Bayes with node discarding

SFAN : selective forest augmented naive Bayes

STAND : selective forest augmented naive Bayes with node discarding

5.2.2 Metrics

LC : local criterion: $log\ p(c_l \mid D)$

SB : standard Bayesian measure with penalty for size

LOO : leave-one-out cross validation

CV10 : ten fold cross validation

CV1010 : ten fold cross validation averaged ten times

5.3 Results

For each candidate set of features, there is a results table below showing the performance of each of 35 possible combinations of classifier and metric. Maximum global performance combinations have been bolded.

	LC	**SB**	**LOO**	**CV10**	**CV1010**
naive	35.065%	35.065%	35.065%	35.065%	35.065%
TAN	42.857%	42.857%	42.857%	42.857%	42.857%
FAN	**48.052%**	38.961%	38.961%	38.961%	38.961%
STAN	23.377%	33.766%	22.078%	22.078%	22.078%
STAND	23.377%	33.766%	22.078%	22.078%	22.078%
SFAN	23.377%	33.766%	22.078%	22.078%	22.078%
SFAND	23.377%	33.766%	22.078%	22.078%	22.078%

Table 3: ALL FEATURES ACTIVE

	LC	**SB**	**LOO**	**CV10**	**CV1010**
naive	35.065%	35.065%	35.065%	35.065%	35.065%
TAN	**48.052%**	**48.052%**	**48.052%**	**48.052%**	**48.052%**
FAN	38.961%	36.364%	36.364%	36.364%	36.364%
STAN	31.169%	29.87%	31.169%	31.169%	31.169%
STAND	31.169%	29.87%	31.169%	31.169%	31.169%
SFAN	31.169%	29.87%	31.169%	31.169%	31.169%
SFAND	31.169%	29.87%	31.169%	31.169%	31.169%

Table 4: EVERYTHING EXCEPT INDIVIDUAL CHARACTER FEATURES

	LC	**SB**	**LOO**	**CV10**	**CV1010**
naive	32.468%	32.468%	32.468%	32.468%	32.468%
TAN	38.961%	38.961%	38.961%	38.961%	38.961%
FAN	38.961%	32.468%	38.961%	35.065%	38.961%
STAN	29.87%	29.87%	29.87%	29.87%	29.87%
STAND	29.87%	29.87%	29.87%	29.87%	29.87%
SFAN	29.87%	29.87%	29.87%	29.87%	29.87%
SFAND	29.87%	29.87%	29.87%	29.87%	29.87%

Table 5: ONLY NLP COMPUTED FEATURES

	LC	**SB**	**LOO**	**CV10**	**CV1010**
naive	31.169%	31.169%	31.169%	31.169%	31.169%
TAN	32.468%	32.468%	32.468%	32.468%	32.468%
FAN	31.169%	31.169%	31.169%	31.169%	31.169%
STAN	31.169%	31.169%	31.169%	31.169%	31.169%
STAND	31.169%	31.169%	31.169%	31.169%	31.169%
SFAN	31.169%	31.169%	31.169%	31.169%	31.169%
SFAND	31.169%	31.169%	31.169%	31.169%	31.169%

Table 6: EVERYTHING BUT NLP COMPUTED FEATURES

	LC	**SB**	**LOO**	**CV10**	**CV1010**
naive	25.974%	25.974%	25.974%	25.974%	25.974%
TAN	31.169%	31.169%	31.169%	31.169%	31.169%
FAN	25.974%	25.974%	25.974%	25.974%	25.974%
STAN	23.377%	33.766%	22.078%	22.078%	22.078%
STAND	23.377%	33.766%	22.078%	22.078%	22.078%
SFAN	23.377%	33.766%	22.078%	22.078%	22.078%
SFAND	23.377%	33.766%	22.078%	22.078%	22.078%

Table 7: ONLY CHARACTER FEATURES

5.4 Analysis

In short, we were satisfied that the minor performance decrement between the MEMM to the NBC was the size we expected it to be. A huge performance drop-off would have indicated that we had an inadequate feature set that was classifier dependant, and was thus not extracting pertinent information about the scripts for training. On the other hand, zero performance drop-off or performance increase would have verified our concerns from earlier, that removing the MEMM's ability to condition on previous decisions actually impaired its performance beyond what can be obtained in a simple generative NBC. The modest decrease in performance (a best F1 of 0.5196 for the MEMM and a best F1 of 0.4805 for the NBC) indicates that, although the MEMM is not conditioning on past decisions, it still outperforms a generative model. It also goes to show that our features generalize across algorithms, leading to effective classifier training in both generative and discriminative models.

We might also attribute a portion of the performance decrement to a limitation in the C4.5 format. For most classification schemes, having only one correct label per datum is appropriate. However, this is not intuitively what we want for our classification task, where movies might belong to several different genres to varying degrees (see PC Score above). Thus, even if the NBC were allowed to report a k-best list like the Viterbi-backed MEMM, C4.5 limits these guesses to be checked against a single, canonical, correct label. This will lead inevitably to lower F1 scores.

6 Conclusions

Given the extensive tests we performed both with feature sets and the two classifiers, we've determined that our classifier and feature sets are behaving as they should. However, the similarity of our accuracy across these datasets is still unnerving and indicative of something else that is wrong.

We believe that our data has an inherent limitation. Half of the labels applied to the 399 scripts were labeled with at least one of Drama, Thriller, Comedy, Action, Crime. Thus, even with a completely wild guess, the classifier still has a pretty good shot at being right if it guesses some of those labels. We did confirm, however, that the classifier makes more-or-less unique guesses for each movie. Thus, we avoid a common problem in classification– believing that our results are accurate when in fact the classifier makes the same guess every time and that guess happens to be correct for the majority of the test data.

Unfortunately, this seems to be the extent of movie scripts freely available on the

Internet. We scoured several free movie script webpages, and they all seem to have a very similar corpus. We grabbed as many as we could, but the size is still about 400. Ideally, we would have had many more data points from a large variety of genres (not mostly the several that we saw).

Despite this limitation, our classifier was still able to perform particularly well in multiway classification. Although the classifier was charged with giving each movie several genre classifications, it was up to 55% accurate at nailing every single genre for a given movie exactly (F1 Score). Considering the relatively small size of our training and test sets, this is a very acceptable performance level for our classifier. With n different labels, a multiway classifier is expected to be correct $\frac{1}{n}$ times, but ours was correct much more frequently.

References

[1] J. Eliashberg et. al, Moviemod: An implementable decision support system for pre-release market evaluation of motion pictures, Marketing Science, Vol. 19, No. 3.

[2] J. R. Finkel, T. Grenager, C. Manning, Incorporating non-local information into information extraction systems by gibbs sampling, Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370.

[3] T. A. S. Foundation, Apache lucene, http://lucene.apache.org/java/docs/index.html.

[4] J. Quinlan, Improved use of continuous attributes in c4.5, Journal of Artificial Intelligence Research,4:77-90.

[5] J. Sacha, jbnc suite, http://jbnc.sourceforge.net.

[6] A. Sadovsky, X. Chen, Song genre and artist classification via supervised learning from lyrics, Previous CS224N Final Project.

[7] M. Santini, Automatic identification of genre in web pages, University of Brighton.

[8] E. Stamatatos et. al, Automatic text categorization in terms of genre and author, Computational Linguistics.

[9] K. Toutanova, C. Manning, Enriching the knowledge sources used in a maximum entropy part-of-speech tagger, In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pp. 63-70.