COMP316 Final Project

Classifying Movie Summaries

Justin Mahabeer, Kaylyn Naidoo, Lwandile Ganyile, Melvin Govender

13 May 2017

Introduction

Text classification is one of the foundational applications of NLP techniques and research. Text Classification is the task of assigning predefined categories to free-text documents. Categorization plays a vital role in many Computer Science applications.

Our project aims to classify movie summaries into their respective genres using a supervised machine learning technique. A movie summary will be taken as a single input to the program and will then be classified into its respective genre(s) using NLP techniques. The general approach is to first train a classifier with labelled-text, that is, movie summaries with a genre already specified, so that unseen movie summaries can be classified based on those labelled (genre specified) movie summaries.

Analysing each Summary

Upon analysing numerous summaries, we were able to determine which techniques needed to be carried out. A portion of a summary is listed below

> ***''In1936, archaeologist Indiana Jones braves an ancient Peruvian temple filled with booby traps to retrieve a golden idol''***

A human will likely be able to categorize the above passage as *Adventure* (or at least not argue that it is likely to be that genre). We thought about WHY we knew Indiana Jones was an adventure and we came to the conclusion that we can simply predict the genre based on keywords in the genre (indicated be the double lines above). This was the theory on which we based our implementation.

Naïve Bayes Classifier

We decided to use the Naive Bayes Classifier due to its simplicity, speed and familiarity to it from lectures. It requires a simple representation of documents (bag of words representation). An important aspect to consider is the fact that Naive Bayes classification gives probabilities for each class (rather than deciding on one class). Movies are usually classified into many genres and so the Naive Bayes Classifier would be most appropriate. The Naïve Bayes classifier is also robust to irrelevant features; irrelevant features cancel each other out.

Corpus

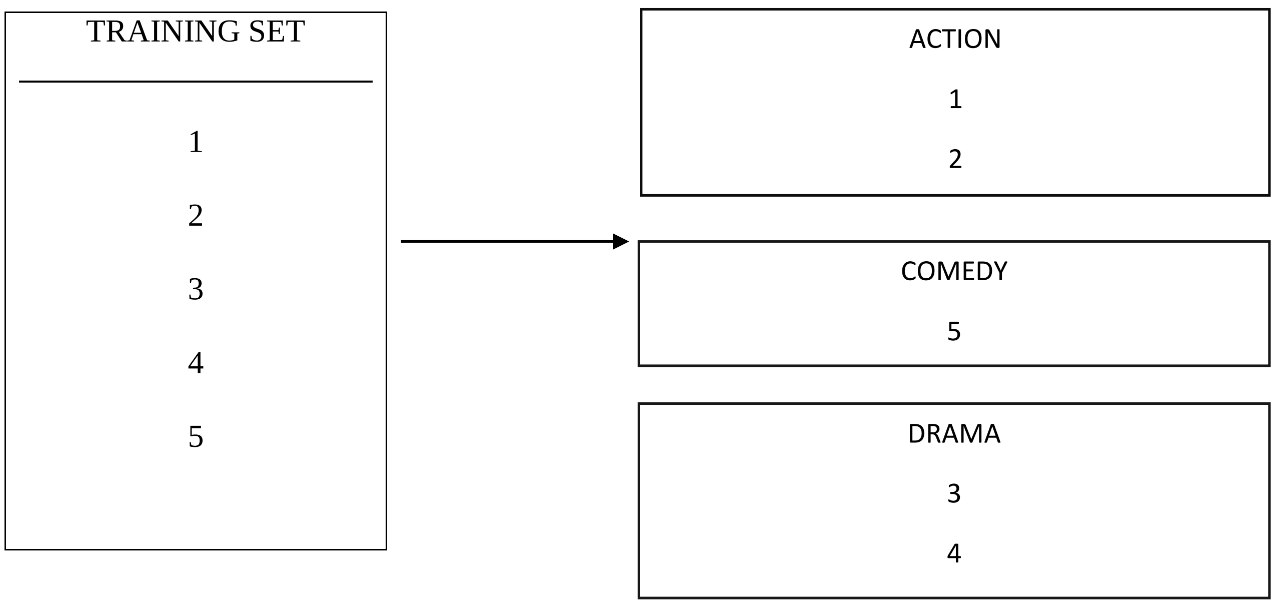
Our corpus of choice is the CMU Movie Corpus which includes a dataset of 42,306 movie plot summaries extracted from Wikipedia + aligned metadata (genre/actor/year released etc.)

The corpus required a bit of cleaning to make processing smoother, movie summaries were in one file and their corresponding genres in another, linking the two was a unique number. This file that contained the genres also contained other information about a movie that wasn't useful to us.

A classifier requires a finite number of classes (genres in this case), so we had to restrict our genres as our corpus had a few genres that were outliers("Erotic/Pornography") and a lot that were just subset of another genre ("Costume Adventure" = "Adventure"). Eventually, we settled on 12 genres(classes). This restriction on genres brought our dataset down to just under 39,000 movie summaries.

Thus, the need for **CorpusClean.java,** which mainly:

1. Restricts the number of genres so we only process those summaries with a relevant genre. Also reformatted the movie.metadata.tsv file, now it only contains a "movie ID" and its corresponding genres.
2. Split the summary corpus into a training set and testing set, then also split the training set according to a movie's genre (to make use of the TF-IDF weighting later on).
3. Before splitting the training set, we removed all the stop words and stemmed all words using the Porter Stemmer algorithm (using Lucene API-Snowball). Removing the stop words is just to process less words when training but stemming will help the machine in determining a feature set and give a truer TF-IDF weighting.



Training/Learning

The main goal for **LuceneTFIDF.java** is to get a feature vector. The feature vector is obtained by indexing through each document (genres, split from training set), tokenizing the text and calculating the TF-IDF weighting for each word, words with a high TF-IDF score are discriminative to this genre, thus should be kept and words with a low TF-IDF score aren't discriminative as they appear in other genres and should be discarded. We would only want words that are discriminative to a particular genre to be included in the feature set, this also drastically reduces the processing time for training, wouldn't want to calculate word probability for words that aren't going to be useful in classifying a movie summary. Bulk of the implementation here is done using the Lucene API.

Once the feature set is obtained, for each word in the feature set, we calculate the probability of that word occurring given a genre, for every genre. Implementation for this is also done in **LuceneTFIDF.java.** We used Laplace smoothing to deal with zero probabilities for words that don't appear in a particular genre. Practically, to avoid underflow from multiplying a lot of probabilities (when classifier calculates the likelihood score), we took the log of probabilities. So instead of multiplying probabilities, we add them to calculate the likelihood score. Since $\log(abc) = \log(a) + \log(b) + \log(c)$, we can just take the log of each word probability now.

The **PriorProbs.java** merely calculates the probability of a particular genre, for all genres. To do this, we had to count the number of times a particular genre occurred in the training set and divide by the total number of genres in the training set. This probability is also used by the classifier to calculate the likelihood score, thus we had to take the log of this probability.

Classifier

The **Classifier.java** is meant to implement a Naïve Bayes Classifier to classify movie summaries into genres from evaluation/test set based on the probabilities gained from our training set. The first step is to get movie summaries from the evaluation set into an appropriate format for processing, it needs to be in a similar format to our training set. That is, removed stop words and stemming. Stemming is particular important as the feature vector contains only stemmed words (because training set was stemmed).

The classifier then needs to calculate the likelihood scores for a particular movie summary for each genre. It simply adds the probabilities (because of the use of the logarithm), probability of a given genre (already calculated from training) and probabilities of each word in the summary occurring in that genre (already calculated from training). The genre that returns the highest likelihood score should be the correct genre for a particular movie summary. There is one case that needs to be handled, a movie summary contains words that our feature set doesn't. This could happen because we reduced the feature set too much or, most likely, because our training set doesn't cover all possibilities. To account for this, the general approach is to assign a really small probability for that word, since we're using the logarithms and just adding probabilities instead of multiplying, we can just assign that word a probability of zero (0).

Evaluation

In terms of evaluation, we worked out the precision, recall and Accuracy values for our results. Our main focus was on a good Precision value. A bad precision value meant we were showing genres which may have had nothing to do with our movie. It would be better for our results to exclude correct genres rather than to include incorrect genres. For example, the movie *Titanic (1997)* has genres listed as Drama and Romance. It would be acceptable for a classifier to only list one of the genres (e.g. Romance) rather than list an incorrect genre (e.g. Drama, Romance, Comedy)

Due to time constraints (and less-than-ideal processing power and memory on our personal machines) we were unable to run our evaluation methods through our entire corpus. However, we have worked collectively on testing the program using movie summaries from our test set as well as online movie summaries (mainly from IMDB). These results were then put together and computed using the following:

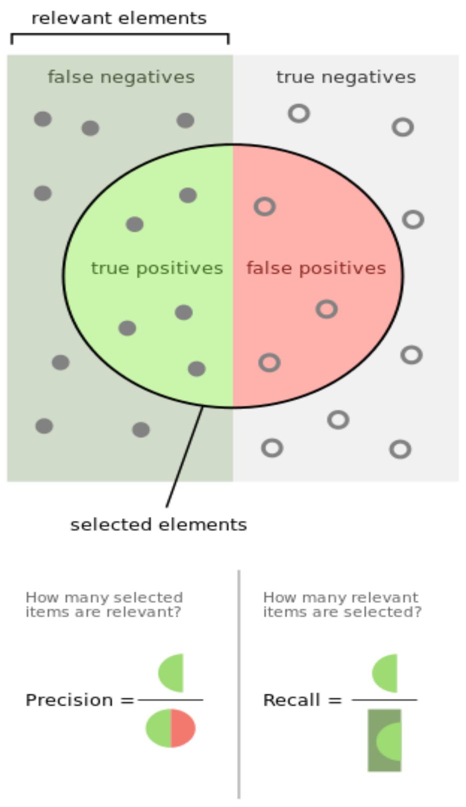
$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Accuracy: 60.5%

Precision (Micro): 62.5%

Recall (Micro): 60.6%



Improvements: Using Cast Members (Actors) as part of the calculations

This would involve adding actor and actress names into their respective summaries. Names and Surnames could be added in the form Dwayne Johnson or Lupita Nyongo depending on which would get through the stemming methods. The reason to do this is that actors tend to play in similar genres throughout their acting careers. This would help with TF-IDF scores. For example, South African Leon Schuster is known for comedies thus Leon Schuster keyword would be added to the comedy file. Our input could take in an optional parameter for cast which would match with the Actor names (Likely to be useful in a real world movie classification)