Movie Genre Classification

CS484 Final Project Report

May 13, 2019

Prepared for: Dr. Jessica Lin

Prepared By: Aziz Amino, Andrew Park

Table of Contents

Abstract

Text processing is becoming increasingly important in today's society. This paper describes the approach that our team took to classify movies based on plot summary, which involved utilizing the python sklearn library and an existing Kaggle movie data set from Wikipedia. The BeautifulSoup library was also used to parse captured HTML pages from Movieweb.com to supplement any holes that were found in the original data set for better performance.

Introduction

The goal of this assignment was to examine various texts of movie plot summaries and determine if this text itself could be examined and used in training a model to classify the genre of other movies based on their plot summaries.

Solution

Pre – Processing

Many of the class labels given within the original dataset had multiple genres concatenated in a variety of ways (back slash, comma, space, etc.). Because many of the movie objects had any number of genre labels (varying between one to three). Because of this, it was decided to train the model with the primary genre. The "primary" genre for the project was defined as the first occurring genre within the label string. Using python's re library, all non – alphabetic as well as non – numeric characters were replaced with a space for ease in splitting the string.

The NLTK parts of speech tagging implementation was used to keep words that would be most valuable in sentiment analysis. Upon observation, this would likely be all forms of descriptor words, all verb and adjective forms. Nouns will typically vary the highest within text (names in particular), expanding the dissimilarity of the texts (retaining stop words would also have this affect).

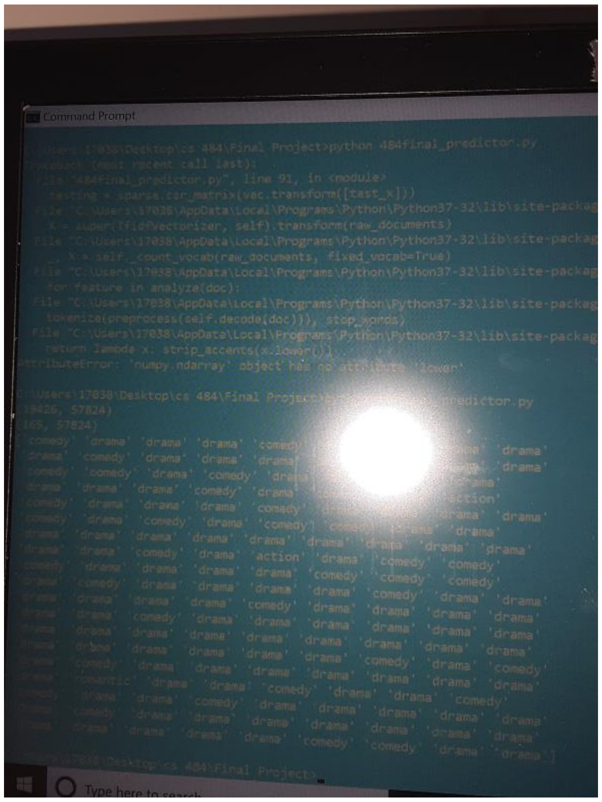
To vectorize the documents, TF-IDF was used in to give more weight to unique terms by multiplying the word frequency within the document by inverse frequency within the document, given more weight to the less frequent words in the vector representations. A main issue with this however, is that the TF-IDF representation build of a bag of words model using term frequency. This form representation does not assign weights by term similarity. For example, in the following two sentences:

Bill has been killed Jane

Jane was murdered by Bill

The most similar words between the two sentences are killed and murdered however, the TF-IDF vectorization will not create representation taking this in to account. Other preprocessor methods that take this account are word2vec and doc2vec. The genism library for these algorithms were experimented (though it became too difficult to get a grasp of the algorithms for implementation by the allocated due date).

The original Kaggle dataset had been severely unbalanced genres comedy and drama outnumbered many genres by thousands while genres such as sci-fi contained only a few hundred (the manner in which the genres labels were extracted as labels for the movie objects may have also affected this imbalance). Because of this, in an early test set sample, the model had severely over predicted for comedy and drama (with only 165 movie objects however still displaying no variance).



```
Command Prompt
[illegible]\Users\17038\Desktop\cs 484\Final Project>python 484final_predictor.py
[illegible] (most recent call last):
  File "484final_predictor.py", line 91, in <module>
    testing = sparse.csr_matrix(vec.transform([test_x]))
  File "C:\Users\17038\AppData\Local\Programs\Python\Python37-32\lib\site-packag
    X = super(TfidfVectorizer, self).transform(raw_documents)
  File "C:\Users\17038\AppData\Local\Programs\Python\Python37-32\lib\site-packag
    _, X = self._count_vocab(raw_documents, fixed_vocab=True)
  File "C:\Users\17038\AppData\Local\Programs\Python\Python37-32\lib\site-packag
    for feature in analyze(doc):
  File "C:\Users\17038\AppData\Local\Programs\Python\Python37-32\lib\site-packag
    tokenize(preprocess(self.decode(doc))), stop_words)
  File "C:\Users\17038\AppData\Local\Programs\Python\Python37-32\lib\site-packag
    return lambda x: strip_accents(x.lower())
AttributeError: 'numpy.ndarray' object has no attribute 'lower'

C:\Users\17038\Desktop\cs 484\Final Project>[illegible]predictor.py
(9426, 57824)
(165, 57824)
['comedy' 'drama' 'drama' 'drama' 'comedy' [illegible] 'drama' 'drama'
 'drama' 'comedy' 'drama' 'drama' 'drama' [illegible] 'drama'
 'comedy' 'comedy' 'drama' 'comedy' 'drama' [illegible] 'drama'
 'drama' 'drama' 'drama' 'comedy' 'drama' [illegible] 'drama'
 'comedy' 'drama' 'drama' 'drama' 'comedy' [illegible] 'action'
 'comedy' 'drama' 'comedy' 'drama' 'comedy' [illegible] 'drama' 'drama'
 'drama' 'drama' 'drama' 'drama' 'drama' 'drama' 'comedy' 'drama' 'drama'
 'drama' 'drama' 'comedy' 'drama' 'action' 'drama' 'drama' 'drama' 'drama'
 'comedy' 'drama' 'drama' 'drama' 'drama' 'drama' 'comedy' 'comedy' 'comedy'
 'drama' 'comedy' 'drama' 'drama' 'drama' 'comedy' 'comedy' 'comedy'
 'drama' 'drama' 'drama' 'drama' 'comedy' 'drama' 'comedy' 'drama' 'drama'
 'drama' 'drama' 'comedy' 'drama' 'drama' 'drama' 'drama' 'drama' 'drama'
 'drama' 'drama' 'drama' 'drama' 'drama' 'drama' 'drama' 'drama' 'drama'
 'drama' 'drama' 'drama' 'drama' 'drama' 'drama' 'drama' 'drama' 'drama'
 'drama' 'comedy' 'drama' 'drama' 'drama' 'drama' 'comedy' 'drama' 'comedy'
 'drama' 'romantic' 'drama' 'drama' 'comedy' 'drama' 'drama' 'drama' 'drama'
 'comedy' 'drama' 'drama' 'comedy' 'drama' 'drama' 'drama' 'comedy'
 'drama' 'comedy' 'drama' 'drama' 'drama' 'drama' 'drama' 'drama' 'drama'
 'drama' 'drama' 'drama' 'drama' 'drama' 'comedy' 'comedy' 'drama' 'drama']

C:\Users\17038\Desktop\cs 484\Final Project>
```

The above image is sample run with for testing 165 movie objects using a decision tree and 20 estimators for bagging. The output displays the predicted labels with an overwhelming majority (possibly all) are predicted as comedy or drama.

A new training set was created aggregating movie instances which resulted in 10,000 entries.

Web Scrapper

The implementation of the web scrapper involved utilizing the BeautifulSoup library, which was used for parsing a requested webpage in HTML format. Movieweb.com was chosen to be scrapped due to the arrangement of movies on this site. Particularly, Movieweb displayed movies in a list arrangement with the relevant data needed for this project, such as title, plot, and genre, when viewed by year.

The web scrapper that was implemented will iterate through every page of results for each year that is specified. On the first iteration for a particular year, the pagination links will be parsed to get the last page of results for that year. This was required for the web scrapper because Movieweb will not return a 404 HTML Error for a URL that is specified with a page that does not exist; Movieweb will simply return the last page in that year's results.

For example, there are only 19 result pages for the year 2019. If a GET request is sent to the Movieweb server with the year 2019 and page 58, Movieweb will return page 19.

Initially, the web scrapper was implemented in a way that wrote all the results for a particular year to a .csv file after the last page of the results page for that year was parsed. However, since Movieweb is structured in the manner described above, the pagination link for the last page would have to be stored for comparison as a way to exit the iteration and write to .csv. Otherwise, the last page would be appended forever with many duplicate entries.

The web scrapper would send a GET request to the Movieweb server with a properly formatted URL based on the page iteration that was going to be parsed. The BeautifulSoup html.parser was then used to extract the a list of movies that were listed on that HTML page. On a 'full' page, there would be a list of twenty movies. This list would then be iterated through to extract the title, plot, and genre for each movie. If the genre for each movie matched one of the genres that our project was classifying for and if the genre was present and if the plot for the movie contained more than 127 characters, the movie would be stored in a pandas DataFrame along with all the other relevant information. However, if the genre did not match or if the genre was not present or if the plot was too short for analysis, the movie would be discarded. For the purposes of this project, the years 1910 to 2020 were scrapped for supplementing the training set. Any duplicate entries were removed in a separate script that aggregated the scrapper data with the previous training data from Kaggle. Any remaining entries that were not used for the training set were used for the test set.

Chosen Models/Ensemble Methods

The models chosen all followed a supervised leaning approach. An attempt to use k-means was used however, being able to extract class labels from the clusters had shown to be more troublesome than applying other models. A multi-layer perceptron was implemented however, the runtime required to receive any results.

Final testing was performed mainly with K – Nearest Neighbor and Decision Tree with some combination of boosting or bagging.

Experiments

Data

the original Kaggle dataset had contain nearly 35,000 movie plot summaries all web scrapped form Wikipedia. The final training set had contained 10,000 movies.

The class labels used are romance, drama, action, comedy, adventure, sci – fi, fantasy, thriller, horror, and animation. Most of the testing was done without animation labeled movies as the genre may cross over heavily with the other class labels due to animation being the form of the story as opposed to the genre.

Experimental Design

As described in the previous section, words that would act as noise (nouns and stopwords) were removed from the text before they had been vectorized by the TF-IDF model.

Because the approach to the problem followed a supervised learning model, metrics that were best suited for rating the correctness of the predictions were accuracy and f1-score. Because f1 – score also utilizes false positives and false negatives along with true positives and true negatives where accuracy only takes into account true positives and true negatives to assign score, f1-score was chosen as the best metric for the various models used.