Final_Report_Team_37

April 29, 2017

1 Final Project Report (CS109b)

1.1 Team 37 (Alexander Dubitskiy, Keenan Venuti, Timur Zambalayev)

Project repository: https://github.com/adubitskiy/cs109b

2 Data Collection

Our task, identifying movie genres based on a self-chosen feature set, utilized two important Python dictionaries for data mining and collection. We used IMDbpy and tmdbsimple which contain methodologies to harvesting information held on IMDb and themoviedb.org, respectively. These datasets hold features such as movie overview, plot, director, actors, crew data, budget, revenue, voting information, reviews, etc. TMDB alone hosts 333,163 movies. For our learning models, we began with a 3k and 5k sub sample before ultimately using a datasets with 10k movies. This dataset became an accumulation of both the IMDB and TMDB movie datasets as they were able to complement each other where some information had not been included. For example, some movies in the TMDB dataset lacked certain revenue, budget and plot data; querying the IMDB system helped account for some discrepancies. When exploring the data, we saw the need to initially try different types of supervised models before making a decision. The datasets are incredibly diverse in potential applications. Both contain hard quantitative data like revenue, budget and average viewer rating, categorical data like actors, directors, crew and textual data such as an overview or plot. Given the stark differences between how these different types of data will contribute to a model, we decided to focus our model's feature sets on relevant predictors that were similar to each other.

3 Data Cleaning

Initially, we toyed around with the datasets, cleaning them, and running small supervised learning classification to get an idea of any potential problems and the limitations of the dataset. We began with a few models based on spending, budget, ratings and quantitative reviews. Here, we began to see some of the problems when classifying movie genres. Firstly, movies fall under multiple genres. One movie may be categorized as just a Drama where as another may be a Drama Comedy. There are a couple of approaches to this type of problem. A data scientist can focus on the genre title that best describes the movie, create hyper-classes via a combination of movie genres and perform multi-class classification, or we can modify the dataset to use multi-labels, where each movie is assigned a vector (the size of the number of all genres) with a binary value if that genre

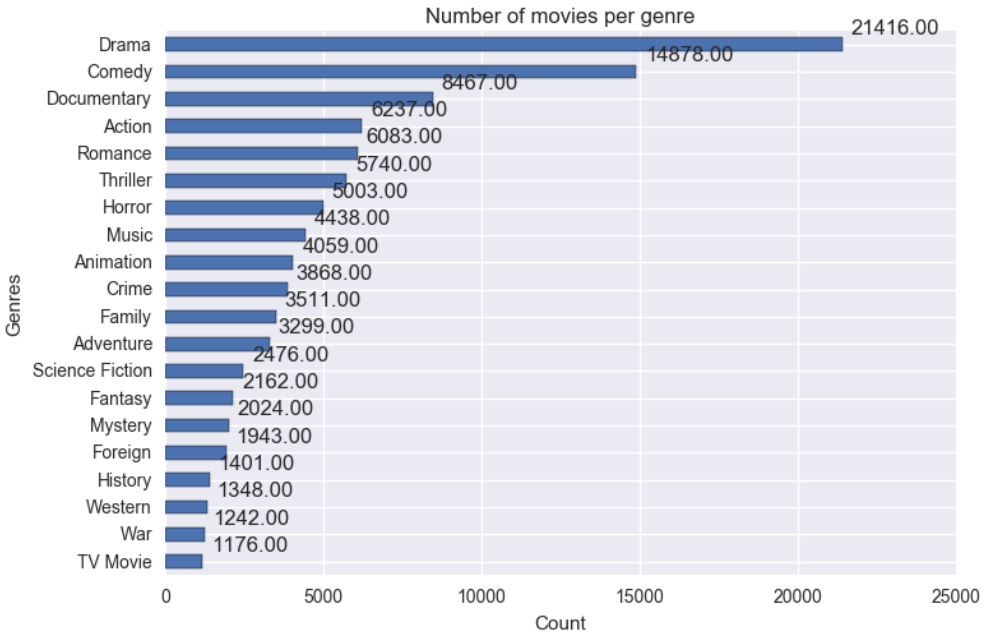

Figure 1. Number of movies per genre

is used in describing the movie. We focused on a multi-label approach as we wanted our model to have strong applicability to the dataset and not make any assumptions. However, we did also utilize a multi-class system when experimenting with neural networks to assist in the network's learning process. After defining our `y` value, we saw the need to select our feature set based on attributes that are similar to each other. But, after creating of this classification `y` set we noticed that there was immense class imbalance in the dataset (see Figure 1). For example, genre is used to define a lot of movies. This will lead to models that over-classify the dominant class in an attempt to increase accuracy hurting applicability. We dealt with this further when selecting our models.

4 Picking Features

One of our initial feature sets focused on director, actor and crew information; this was comprised of around X features. Movies often contain the same actors playing the same role and directors almost always stay within a genre. But, we also saw predictive power in the overview/plot descriptions within the dataset. From there we implemented another model using overviews as a feature set via vectorization of the descriptions through bag of words. This process involves taking all the overviews in the dataset, eliminating stop words and creating a vector, sized at this new "bag of words". Then, each overview is represented as a vector based on which words were used. Eventually, we saw that subseting the data based on features created accuracy weaknesses. We combined our features sets into one more complete dataset and looked to picking a supervised learning model. Experimenting with Supervised Learning Models We trained several classification models including a LDA, QDA, random forest, tree and SVM (linear and radial kernels). Ultimately, we chose a random forest and linear support vector machine with a stochastic gradient decent learning method (SGD). We found that these models were well suited for our two types of datasets (director/actor/crew and overviews). They were also different enough to properly express the data. Specifically, our SGD model cannot natively support multi-label classi-

fication. Instead, we used a one vs rest classification method within the model. This fits a different model for every class in the multi-label y set. Conversely, the random forest can natively handle multi-label data. The SGD model is based on a linear SVM, so it gains some of the qualities of that model. It can train very quickly and performed decently when compared to the random forest. The random forest takes more time as it fits specific curvatures of the train data. With the selection of these two very different models and their application to our combined dataset datasets, we focused on tuning.

5 Tuning

We tuned both our SGD and random forest model using an sklearn function, GridsearchCV. This takes in parameters to tune models on and finds the best combination to implement in the final model. After testing different alpha estimators through cross validation for our SGD model, we found an optimal value and created the final model. The random forest model was tuned on max features and the minimum sample leaf via grid-search. Both models were tuned based on their hamming loss score.

6 Dummy Classifiers

To better understand our model's performance, we implemented three different dummy classifiers. Using these comparisons, we could determine if our models had applicability or were as helpful as guessing a certain way. Within our dummy classifiers, the stratified classifier is a classification algorithm that takes the training set class distribution and assigns classes to the test set based on that distribution. The uniform dummy classifier assigns classes in uniform to all observations in the testing set. Finally, the most frequent dummy classifier assigns the most frequent class to every observation. However, in this context, our classes are multi-labels.

7 Performance of Models

We compared accuracy across genres of both our SGD and Random Forest model against each other and the dummy classifiers. Choosing the right metric is a challenge within itself as some may be misleading while others are not relevant. For example, our classifier that assigned every observation to the most dominant class had the smallest hamming loss, but a model like this is very unhelpful. We decided to use the f-1score as a our most important metric to understand a model's accuracy; it incorporates both precision and recall (alone these metrics can be deceiving) to give a fuller understanding of a model's strengths and weaknesses. Below is the performance of each model:

7.1 need to re-run to get averages

- SGD Average Precision: Average Recall: Average f-1 score:
- Random Forest Average Precision: Average Recall: Average f-1 score:
- Dummy Models Uniform Classifier Average Precision: .21 Average Recall: .51 Average f-1 score: .27

7.2 need to re-run

Most popular classifier Average Precision: Average Recall: Average f-1 score: Stratified Classifier Average Precision: .21 Average Recall: .21 Average f-1 score: .21