Movies' Genres Classification by Synopsis

Ka-Wing Ho

Abstract

This paper investigated four different approaches for automated movies' genres classification based on their synopsis.

1 Introduction

Public movies' database such as IMDB provides genre information to assist searching. The tagging of movies' genres is still a manual process which involves the collection of users' suggestions sent to known email addresses of IMDB. Movies are often registered with inaccurate genres. Automatic genres classification of a movie based on its synopsis not only speeds up the classification process by providing a list of suggestion but the result may potentially be more accurate than an untrained human.

2 Data set

The data used in this project is a set of text files from IMDB [1]. They contain 158,840 synopsis [4] and 746,883 pieces of genre information [3] of movies and TV shows. For this project, **16,000** *unique* titles at which both the synopsis and genre information were available were chosen and randomly split into 80% and 20% sets. The former was used for training while the latter for testing. Note that a movie can be (and often so) associated with **more than one** genres. There are 20 listed genres in the IMDB data set and only the **10** (denoted as L) most common genres were used in this project. The genres' names and percentages of movies in them are: *action(*13.2%*), adventure(*9.1%*), comedy(*30.0%*), crime(*13.1%*), documentary(*16.3%*), drama(*49.1%*), family(*11.0%*), romance(*15.65%*), short films(*33.4%*) and thriller(*13.6%*)* respectively.

The synopsis text file was processed to generate the *bag-of-words (BoW)* representation. The NLTK stopwords list [2] was first used to remove all stopwords from the synopsis. All numerical words were also mapped to the same index in the dictionary as they usually don't provide much genre related information. The remaining words were filtered through a Python stemmer library [5] so words with the same base but different forms mapped to the same index in the dictionary. For the 16,000 titles used in this project, a dictionary with 63,840 words was generated. Out of the 63,840 words, only those which have occurred more than 15 times in all the *training samples* were used in BoW representation so only **10,656** (denoted as V) words were left. The term frequency inverse document frequency (*tfidf*) of the words were then computed:

$$tfidf(w_k, d_i) = w_{ki} * log\frac{m}{\sum_{d=1}^{m} 1\{w_{kd} > 0\}}$$

where w_{ki} is the frequncy of the k-th word in the i-th movie's synopsis (d_i), and m is the number of training/test sets.

After the preprocessing above, each movie's synopsis was represented by its *tfidf* vector $x \in \mathbb{R}^{10,656}$. The vector was also normalized to make it a *unit vector*. This was done to account for the variation in length between the synopsis of different movies. Similar treatment of *tfidf* and normalization were also done for the test set using the same 10,656 words. There is also a bit-vector $y \in \mathbb{R}^{10}$ associated with each movie where $y_l = \{0, 1\}$ to indicate whether it belongs to genre l.

3 Classification Methods

A movie can be associated with multiple genres so the task in this project is a mult-label classification problem. We evaluated four different approaches, some of which were published in previous literature:

- One-Vs-All approach with SVM
- Multi-label K-nearest neighbor (KNN) [7]
- Parametric mixture model (PMM) [8, 9]
- Neural network

3.1 SVM

While a movie can belong to mutliple genres, whether tagging it wih a paritcular genre is just a binary classification problem. Specifically, one can group all movies of a particular genre together as the positive samples and the rest as negative samples and train a binary classifier with these two disjoint sets. This approach is generally known as One-Vs-All. However, if the sizes of the two classes differ a lot, the classifier resulted is likely to bias towards the more populous class. While the precision may be good, it would result in a rather low recall for less common genres (e.g. *adventure*). So, we also tried two different approaches: (1) increase the weight (option -w in libsvm [6]) of the penalty for misclassifying the smaller class. One simple way to determine the weight is based on the ratio between the sizes of the two classes; (2) reduce the training set of the more populous class to match the size of the less populous class. We used libsvm [6] in this project. The input is the normalized *tfidf* matrix and the corresponding output labels for each genres. We tried different solvers and misclassification penalty and found that the combination of *L2-regularized L1-loss support vector classification (dual)* with default penalty of 1 yielded the best result.

3.2 Multi-label K-nearest neighbor

Intuitively, movies belonging to the same genre should share more common keywords in their synopsis. If one were to consider a movie synopsis x as a point in the hyperspace, movies with similar genres combination of x should be close to it. This is similar to the idea behind multi-label KNN alogrithm in [7] which utitlizes the statisical information of a document's k-nearest neighbor to infer its genres.

For a training sample ($x^{(i)}, y^{(i)}$), its k-nearest neighbors are denoted as $N^{(i)}$. Let $C_l^{(i)} := \sum_{a \in N^{(i)}} y_l^{(a)}$ denote the number of its k-nearest neighbors which belong to the genre l. Let E_l^j be the event that $C_l^{(i)} = j \in \{0, 1, ..., k\}$ (i.e. j out of k neighbors are in genre l) and H_l^p be the event that $y_l^{(i)} = p \in \{0, 1\}$. The task of classifying whether a test sample t belongs to genre l is by computing the maximum a posteriori(MAP) estimate:

$$\begin{aligned} y_l^{(t)} &:= \arg\max_{p \in \{0,1\}} P(H_l^p | E_l^{C_l^{(t)}}) \\ &= \arg\max_{p \in \{0,1\}} \frac{P(H_l^p) P(E_l^{C_l^{(t)}} | H_l^p)}{P(E_l^{C_l^{(t)}})} \\ &= \arg\max_{p \in \{0,1\}} P(H_l^p) P(E_l^{C_l^{(t)}} | H_l^p) \end{aligned}$$

$P(E_l^j | H_l^p)$ is estimated from the training set by updating counters $d_l^0[j]$ and $d_l^1[j]$ as follows:

```
$d_l^0[j] := 0;\ d_l^1[j] := 0; \forall\, j \in \{0, ..., k\}$
for $i \in \{1, ..., m\}$
    for $l \in \{drama, comedy, thriller, ...\}$
        if $(y_l^{(i)} == 1)$
            $d_l^1[C_l^{(i)}]$++;
        else
            $d_l^0[C_l^{(i)}]$++;
        end
    end
end
```

With Laplace smoothing,

$$P(E_l^j | H_l^p) = \frac{d_l^p[j] + 1}{\sum_{j'=0}^{k} d_l^p[j'] + k + 1}, \forall\, l, j, p$$

No smoothing is needed for the prior as there is at least a movie in each genre:

$$P(H_l^p) = \frac{\sum_{i=1}^{m} 1\{y_l^{(i)} = p\}}{m}, \forall\, l, p$$

Alternately, one can also assume that the prior follows a uniform distribution. In this case, it's the same as computing the maximum likelihood estimate(MLE):

$$y_l^{(t)} := \arg\max_{p \in \{0,1\}} P(E_l^{C_l^{(t)}} | H_l^p)$$

3.3 Parametric Mixture Model

In [8, 9], the authors argued that the binary classification approach taken in section 3.1 doesn't represent the the multi-faceted negative samples well. They proposed a generative model which conjectured that a document belonging to muliple categories can be regarded as a mixture of words related to those categories. Specifically, for a word w, suppose its probability of showing up in the synopsis of a movie in genre l is the parameter $\theta_{l,w} = P(w|l)$ where $\sum_{w=1}^{V} \theta_{l,w} = 1$. For a given movie belonging to multiple genres, the probability of the word w showing up in its synopsis is a weighted sum of the word distribution over each genres: $P(w|y^{(i)}) = \sum_{l=1}^{L} \lambda_l^{(i)} * \theta_{l,w}$ where $\sum_{l=1}^{L} \lambda_l^{(i)} = 1$ and $\lambda_l^{(i)} = 0$ if $y_l^{(i)} = 0$. We implemented the *PMM1 model* in [9] which made a simplifying assumption that $\lambda_l^{(i)}$ has a uniform distribution over all the genres to which a movie belongs. In other words, $\lambda_l^{(i)} := y_l^{(i)} / \sum_{l=1}^{L} y_l^{(i)}$. For example, if a movie is an *action drama,* its $y^{(i)} = [1,0,0,0,0,1,0,0,0,0]$, then $P(w|y^{(i)}) = 0.5 * \theta_{1,w} + 0.5 * \theta_{6,w}$.

With the naive Bayes assumption, the probability of a synopsis $x^{(i)}$ given genres $y^{(i)}$ is:

$$P(x^{(i)}|y^{(i)}, \Theta) = \prod_{i=1}^{V} (\frac{\sum_{l=1}^{L} y_l \theta_{l,w}}{\sum_{l'=1}^{L} y_{l'}})^{n_w^{(i)}}$$

where $n_w^{(i)}$ is the frequency of word w in $x^{(i)}$ and Θ is the set of all $\theta_{l,w}$. Θ is estimated by maximizing the posterior $P(\Theta|\Omega)$ where $\Omega = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$. Assume y is independent of Θ. Taking log on both sides gives $\Theta_{map} := \arg\max_{\Theta} \sum_{i=1}^{m} \{logP(x^{(i)}|y^{(i)}, \Theta) + logP(\Theta)\}$ where the prior over the parameters is assumed to be $P(\Theta) \propto \prod_{l=1}^{L} \prod_{w=1}^{V} \theta_{l,w}$ in [9]. So, the objective function to maximize is:

$$J(\Theta) = \sum_{i=1}^{m} \sum_{w=1}^{V} n_w^{(i)} log \sum_{l=1}^{L} \lambda_l^{(i)} \theta_{l,w} + \sum_{l=1}^{L} \sum_{w=1}^{V} log\theta_{l,w}$$

The authors in [9] proved that the objective function can be maximized by iteratively updating the followings until convergence:

$$g_{l,w}^{(i)}(\Theta) := \frac{\lambda_l^{(i)} \theta_{l,w}}{\sum_{l=1}^{L} \lambda_l^{(i)} \theta_{l,w}}$$

$$\theta_{l,w}^{(t+1)} := \frac{\sum_{i=1}^{m} n_w^{(i)} g_{l,w}^{(i)}(\Theta^{(t)}) + 1}{\sum_{w=1}^{V} \sum_{i=1}^{m} n_w^{(i)} g_{l,w}^{(i)}(\Theta^{(t)}) + V}, \ \forall l, w$$

For a given test sample t, prediction of its genres $y^{(t)}$ is:

$$y^{(t)} := \arg\max_{y'} P(x^{(t)}|y'; \Theta_{map}) P(y')$$

where prior $P(y')$ can be estimated from the training set. This is essentially computing the maximum a posteriori (MAP) estimate. Note there are $2^{10} - 1$ possible values which y' can take on, each corresponding to different combination of the 10 genres. Alternately, one can once again assume a uniform class prior as in [9]. This is essentially the MLE. We experimented with both in this project.

3.4 Neural network

Lastly, we also trained a neural network using error backpropagation. The neural net consisted of one hidden layer, an input layer of size V(10,656) and output layer size of L(10). 2 different hidden layer sizes (S) were tried: 100 and 300. The activation function was the sigmoid function and the cost function with regularization was:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{l=1}^{L} [-y_l^{(i)} log(h_\theta(x^{(i)})_l) - (1-y_l^{(i)}) log(1-h_\theta(x^{(i)})_l)]$$

$$+ \frac{\lambda}{2m} [\sum_{j=1}^{S} \sum_{k=1}^{V} (\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{L} \sum_{k=1}^{S} (\Theta_{j,k}^{(2)})^2]$$

where $h_\theta(x^{(i)})_l$ is the output of l-th output unit, $\Theta^{(1)} \in \mathbb{R}^{SxV}$ and $\Theta^{(2)} \in \mathbb{R}^{LxS}$ are the weights for the two connecting layers.

We also attempted to utilize PCA to reduce the dimension of the input data. We used the *princomp* command in Matlab to generate the principal components and the mappings of the *tfidf* vectors to the space of the principal components and trained neural networks with different number of principal components. The test set was also mapped into the space of the principal components before being fed into the trained neural network to make prediction.

4 Results

	Precision	Recall	**F Measure**
SVM(Default)	0.66141	0.39572	**0.47151**
SVM(weighted)	0.47689	0.62533	**0.53785**
SVM(1:1)	0.51205	0.61631	**0.54999**
KNN(k=97)(MLE)	0.40987	0.73400	**0.51580**
KNN(k=88)(MAP)	0.64093	0.33261	**0.40060**
PMM(MLE)	0.46371	0.54234	**0.48550**
PMM(MAP)	0.53624	0.45876	**0.47375**
NN($\lambda = 1, PC = 1000$)	0.67630	0.41513	**0.49896**
NN($\lambda = 1, PC = 4000$)	0.65444	0.43225	**0.50849**
NN($\lambda = 1, PC = 8000$)	0.62493	0.45708	**0.52044**
NN($\lambda = 1$, No PCA)	0.64453	0.43666	**0.50938**
NN($\lambda = 0$, No PCA)	0.66886	0.41655	**0.49786**

Table 1: Average precision, average recall and average F-Measures over all genres.

The evaluation is based on the following metrics:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

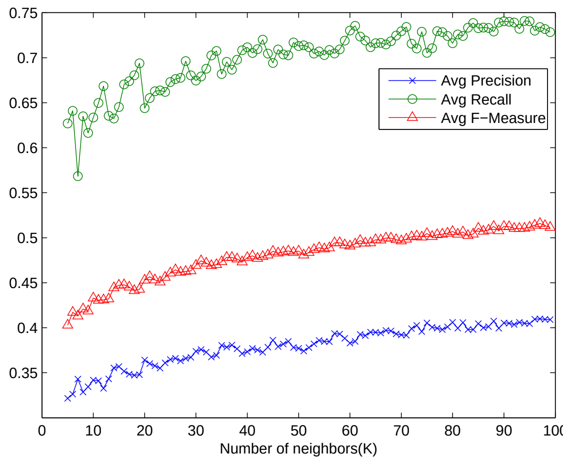

Figure 1: KNN(MLE) over different K

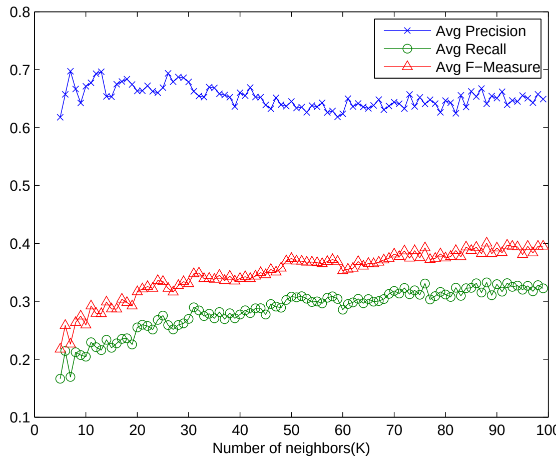

Figure 2: KNN(MAP) over different K

$$F\ Measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

where TP, FP and FN are the numbers of true positive, false positive and false negative results respectively. Table 1 shows the average precision, average recall and average F-Measures over all genres by different algorithms. Table 2 shows the precision and recall information for each genres.

SVM(Default) and *SVM(weighted)* were SVM classifiers trained with the entire training set but the latter assigned more weight to the misclassification penalty of the smaller class; *SVM(1:1)* was a SVM classifier trained with a subset of the training set so that sizes of both classes were similar. *KNN(MLE/MAP)* and *PMM(MLE/MAP)* were the K-nearest neighbor and parametric mixture model respectively; *NN(*)* stand for various configurations of a neural network with 100 hidden units: λ is the weight of the regularization term and PC is the number of principal components used if PCA was applied to the inputs.

As shown in table 2, *SVM(Default)* had a low recall rate for the less common genres in the data set as the imbalance in class sizes resulted in a decision boundary which favored the negative class. There is also a notable difference in the recall between MLE and MAP for the *KNN* model and, to a lesser extent, the *PMM* model. For less common genres, its prior $P(H_l^1)$ is much less than $P(H_l^0)$ so the posterior probability $P(H_l^1|E_l^{C_l^{(t)}})$ was dragged down by it, resulting in lower recall (but higher precision) than *KNN(MLE)*. Similar effect was observed in the *PMM(MAP)* model but the difference with *PMM(MLE)* was not as much. There were $2^{10} - 1$ possible priors to consider so the skewness among them was more evenly “spread out”.

We experimented with values of K from 5 to 99 for the *KNN* models. Figure 1 and Figure 2 show the average precision, recall and F measure over different numbers of K. Interestingly, for *KNN(MLE)*, both the precision and recall kept rising as the number of neighbors increased. This may suggest that movies with the same genres weren’t as closely clustered as initially thought. It may also be the case that different genres perform best at different values of K.

As mentioned in section 3.4, we tried neural networks with hidden layer sizes of 100 and 300 respectively. The latter actually had a slightly inferior performance to the former. The high dimensionality of the input layer ($V > 10,000$) led to huge number of parameters in the weight matrix $\Theta^{(1)}$ relative to the number of training samples($12,000+$). A neural network with fewer numbers of hidden units has fewer parameters and it may be less prone to overfitting in this case. We attempted to reduce the the input dimension with PCA but it appears that PCA may not be as effective on this data set as one would have desired: the maximum amount of variance captured by a single principal component was only 0.38%. The first 1000, 2000, 4000 and 8,000 principal components accounted for about 46%, 66%, 86% and 98% of the variance respectively. It’s interesting to note a neural network trained with only the first 1000 principal components (i.e. 10% of the size of the original input layer) still managed to achieve similar level of precision and recall as one trained with all components (albeit without regularization).

	action		*adventure*		*comedy*		*crime*		*documentary*	
	p	**r**	**p**	**r**	**p**	**r**	**p**	**r**	**p**	**r**
SVM(Default)	0.67978	0.27945	0.64516	0.12903	0.68563	0.48638	0.65165	0.48984	0.77209	0.64591
SVM(weighted)	0.39970	0.61201	0.32088	0.47097	0.56616	0.63471	0.46769	0.68623	0.63636	0.80350
SVM(1:1)	0.31752	0.75751	0.23455	0.72258	0.55261	0.69425	0.41002	0.81264	0.54177	0.88327
KNN(k=97)(MLE)	0.34289	0.71824	0.24741	0.61613	0.52611	0.68113	0.40883	0.81490	0.49607	0.85992
KNN(k=88)(MAP)	0.68116	0.21709	0.62000	0.10000	0.62623	0.38547	0.66803	0.36795	0.70110	0.62062
PMM(MLE)	0.47079	0.31640	0.34642	0.48387	0.61950	0.52573	0.49094	0.73363	0.40304	0.87743
PMM(MAP)	0.59146	0.22402	0.46311	0.36452	0.63568	0.42785	0.57787	0.63657	0.48876	0.80350
NN($\lambda = 1, PC = 1000$)	0.65700	0.31409	0.69136	0.18065	0.68188	0.48234	0.73592	0.47178	0.80446	0.63230
NN($\lambda = 1, PC = 4000$)	0.62869	0.34411	0.63810	0.21613	0.64520	0.51564	0.71525	0.47630	0.80676	0.64981
NN($\lambda = 1, PC = 8000$)	0.60886	0.38106	0.57031	0.23548	0.61341	0.50757	0.69159	0.50113	0.79350	0.66537
NN($\lambda = 1$, No PCA)	0.62705	0.35335	0.59091	0.20968	0.64885	0.51463	0.71380	0.47856	0.80328	0.66732
NN($\lambda = 0$, No PCA)	0.65385	0.31409	0.64444	0.18710	0.65409	0.52472	0.74060	0.44470	0.80332	0.65953
	drama		*family*		*romance*		*short films*		*thriller*	
	p	**r**	**p**	**r**	**p**	**r**	**p**	**r**	**p**	**r**
SVM(Default)	0.69486	0.73624	0.61607	0.19828	0.60465	0.26971	0.70474	0.54460	0.55944	0.17778
SVM(weighted)	0.69444	0.73624	0.31707	0.52299	0.40112	0.59336	0.61793	0.66667	0.34751	0.52667
SVM(1:1)	0.69337	0.73688	0.22743	0.73851	0.34342	0.75311	0.60235	0.72113	0.31082	0.74667
KNN(k=97)(MLE)	0.65719	0.76953	0.25000	0.64943	0.32866	0.73029	0.56079	0.74930	0.28073	0.75111
KNN(k=88)(MAP)	0.66292	0.75416	0.70968	0.12644	0.62838	0.19295	0.66917	0.50141	0.44262	0.06000
PMM(MLE)	0.68331	0.58707	0.33191	0.44828	0.41344	0.33195	0.48695	0.66573	0.39080	0.45333
PMM(MAP)	0.68304	0.63188	0.49785	0.33333	0.44231	0.19087	0.49341	0.63286	0.48889	0.34222
NN($\lambda = 1, PC = 1000$)	0.71704	0.73816	0.62416	0.26724	0.62025	0.30498	0.70631	0.54648	0.52459	0.21333
NN($\lambda = 1, PC = 4000$)	0.68118	0.71127	0.61875	0.28448	0.61134	0.31328	0.66027	0.54930	0.53881	0.26222
NN($\lambda = 1, PC = 8000$)	0.69135	0.71127	0.56784	0.32471	0.57333	0.35685	0.65396	0.59624	0.48519	0.29111
NN($\lambda = 1$, No PCA)	0.69212	0.70807	0.59756	0.28161	0.57664	0.32780	0.67822	0.58779	0.51691	0.23778
NN($\lambda = 0$, No PCA)	0.70633	0.70679	0.62774	0.24713	0.60956	0.31743	0.71377	0.55962	0.53488	0.20444

Table 2: Precision (**p**) and recall (**r**) of each genres by different algorithms.

References

[1] Imdb alternative interfaces. URL `http://www.imdb.com/interfaces`.

[2] Natural language toolkit. URL `http://www.nltk.org`.

[3] Imdb genres database, 2011. URL `ftp://ftp.fu-berlin.de/pub/misc/movies/database/genres.list.gz`.

[4] Imdb plots database, 2011. URL `ftp://ftp.fu-berlin.de/pub/misc/movies/database/plot.list.gz`.

[5] R. Boulton. Pystemmer 1.0.1. URL `http://pypi.python.org/pypi/PyStemmer/1.0.1`.

[6] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines, 2001.

[7] M. ling Zhang and Z. hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40:2038–3048, 2007.

[8] A. K. McCallum. Multi-label text classification with a mixture model trained by em. In *AAAI 99 Workshop on Text Learning*, 1999.

[9] N. Ueda and K. Saito. Parametric mixture models for multi-labeled text. In *In Advances in Neural Information Processing Systems 15*, pages 721–728. MIT Press, 2003.