Predicting Multi-label Movie Genres From Plot Summaries

Sina AKHAVAN FARSHCHI - Armita KHAJEH NASSIRI

***Abstract*—This project aims to predict movie genres based on plot summaries. Text understanding starts with the challenge of finding machine-understandable representation that captures the semantics of texts. In this project, we used three of the most commonly used document representations to represent movie plots. To tackle the multi-label classification problem of predicting movie genres, we used methods such as Binary Relevance, Classifier chains, Decision Tree Classifier, and BPMLL. Experiments have been conducted and several evaluation metrics have been used to report the results.**

I. Introduction

With the recent surge of research on text classification, a considerable amount of attention has been drawn to both text representation and Multi-label Classification. For instance, a typical important problem of this kind is classifying news articles to different topics based on their context. One must note that a piece of news may fall into several categories at the same time and thus it is multi-labeled. The data we worked on in this project was The Internet Movie Database (**IMDB**)[1] which contained movie names, movie summaries and genres. In this dataset each instance of movies can be associated with a subset of labels. It consists of 301134 rows, each row being a particular movie, and 28 different genres. There exists 9897 different combinations of labels and each movie belongs on average to 2.17 genres. It's worth mentioning that the dataset has very imbalanced classes. The distribution of genres can be seen in figure1. In section II we will first briefly explain the methods we used for achieving Vector representation of our documents (plot summaries). These vectors are needed so as to be given as input to the chosen classification method. Section III outlines the different multi-label classification methods we investigated. Section IV defines the metrics we reported our results on and finally in section V we plot and compare the experimental results for different methods. Section VI summarizes the paper.

II. Vector Representation

In order to predict what genres a movie belongs to based on plot summaries, it is crucial to first come up with an effective document representation model for plot summaries to become machine understandable. In literature, several methods have been suggested. In this paper, we investigated three different models: Bag of words, Word2Vec, and Doc2Vec. It should be mentioned that in the pre-processing step, the English stop words have been removed, Part of speech tagging has been done and only adjectives, nouns and verbs have been kept. Extra spaces and punctuations have been omitted too and the plots have been tokenized. The preprocessing step took 1392 seconds. In what follows we are going to briefly explain each of these methods and in the end compare their execution time in Table I.

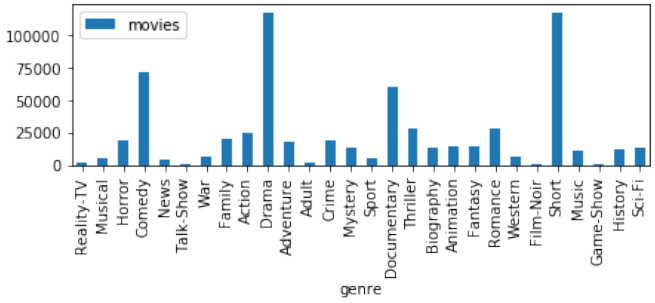

Fig. 1: Distribution of genres **IMDB** dataset

A. Bag-of-Words

Bag-of-words (BoW) is one of the basic and most commonly used methods for document representation due to its simplicity. It is essentially an algorithm that forms a vector by counting how many times each word appears in the document. It is needless to mention that Bag-of-words suffers from several limitations, for instance, it fails to capture relationships between words since it deals with each word individually. It also suffers from high dimensionality of representation and sparsity. A simple twist to BoW would be Tf-idf. Tf-idf stands for term frequency-inverse document frequence. Tf-idf captures the importance of a word in a document so unlike BoW, it doesn't emphasize a word more than it's needed. The Tf-idf score (weight) of term t in plot p is given by: $\mathbf{w_{i,p}} = \mathbf{tf_{i,p}} \times \mathbf{log}\frac{\mathbf{N}}{\mathbf{df_i}}$ where $\mathbf{tf_{i,p}}$ is the number of times i-th term has appeared in plot p. Since the naive representation of $\mathbf{tf_{i,p}}$ causes bias towards long plots, as a given term has more chance to appear in longer plot, all $\mathbf{w_p}$ values are normalized as $\|\mathbf{w_p}\| = 1$. df_i is the number of plots containing the i-th term and $\mathbf{N}$ is the total number of plots. We have also tried Tf-idf as a more sophisticated version of BoW.

B. Word2Vec

Word2Vec is an efficient algorithm proposed in [13] by Mikolov et al. in 2013. It is a shallow (two-layer) neural net that learns embeddings for words and essentially outputs vectorized representations for words (every vector is mapped to a unique vector). What is really interesting about Word2Vec is that it is able to capture and encode many linguistic regularities and patterns. For instance, as mentioned in the original

[1] available at https://www.dropbox.com/s/d4gdx0yewvq5sne/imdb.csv.gz?dl=0

paper, semantic relationships are often preserved in vector operations on word vectors, e.g., vec(Berlin) - vec(Germany) + vec(France) is close to vec(Paris). This means that distances between embedded word vectors are to some extend meaningful. There are two model architectures in Word2Vec to learn word embeddings: continuous bag-of-words (CBOW) and continuous skip-gram. In CBOW, the model predicts the target word from a window of surrounding context words. In the continuous skip-gram architecture, the model predicts the surrounding window of context words using the current word.

In the two-layer neural network architecture of Word2Vec, we have an input layer, a projection layer and an output layer. Let $\mathbf{U}$ be the matrix the projection layer is parametrized by and let $\mathbf{V^T}$ be that of the output layer. The probability of observing the word $\mathbf{w}$ in its context $\mathbf{c}$ in document D is defined using the Softmax function as:

$$P(w|c) = \frac{\mathbf{exp}(V_w^T \mathbf{U} c)}{\sum_{w \in V} \mathbf{exp}(V_w^T \mathbf{U} c)} \quad (1)$$

Then the objective is to learn the word embeddings such that they maximize the log likelihood of observing the word we want to have vectorized at each position of the document. We observe the words in a context, the size of this context is a parameter that has to be chosen. The larger the size of training context, the higher the accuracy since we observe more words and more examples are trained. However, this comes at the expense of training time.

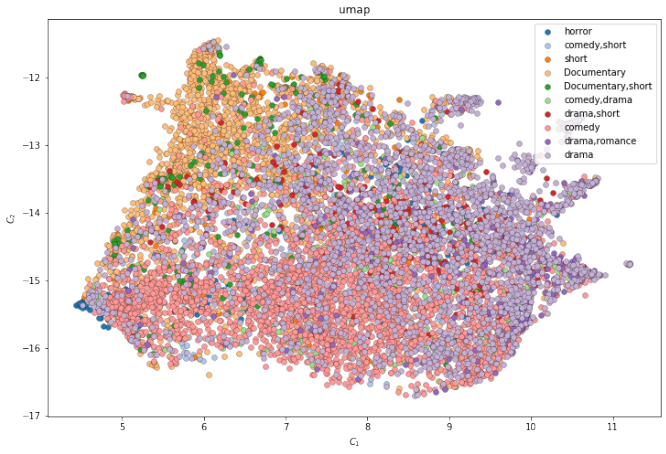

Fig. 2: Low dimensional representation of average Word2Vec embedding vectors for plots having labels corresponding to 10-most frequent labels with UMAP

In the scope of this project we decided to use Google's pre-trained Word2Vec model [1] in which the vector length is set to 300. It includes word embeddings for a vocabulary of about three million words that have been trained on Google News dataset. It's roughly 3.5 GB.

What we actually need, is a single vector that would represent each plot rather than representing words in the vocabulary. Thus one natural way to achieve a vector representation for each plot from Word2Vec, is to get the average of the word vectors for all words in each plot. However, Mikolov et al. in [14] argued that averaging word vectors for each document, loses the word order in the same way as the standard bag-of-words models do in other words, it fails to capture the sequential relationship. However, according to [12], simply averaging word embeddings of all words in a text has proven to be a strong baseline or feature across a multitude of tasks. We have employed this approach in this project to achieve a vectorized representation for each plot. A more sophisticated variant that can be investigated as future work is to weight word vectors with their corresponding Tf-idf to decrease the influence of the most common words.

C. Doc2Vec

Doc2Vec is an unsupervised algorithm that was first proposed for paragraph level embedding in [14] by the same research team responsible for Word2Vec and was named Paragraph Vector in the original paper. The idea and math behind Doc2Vec is very similar to that of Word2Vec and the exact same procedure holds for word embedding training. It can be said that Doc2vec is the same as Word2Vec but in the context of documents. The only difference is that Doc2Vec introduces an extra global vector to the context to represent the document the window is in. Furthermore, Doc2Vec forces the extra global vectors to learn the information that is missing from the local context. This is necessary and helpful for performing predictions in the global document. The extra global vector acts as a memory that can remember what is missing from the current context and can itself be thought of as another word. We employed Doc2Vec to get vectorized representation for each plot using Gensim library and we set the fixed length for vectors to be 300.

TABLE I: Learning time for different representation learning algorithms discussed in II

document representations	**Learning time-seconds**
Bag-of-Words	0.09
Tf-idf	4.84
Word2Vec	551.34
Doc2Vec	3856.86

III. Multi-Label Classification

Multi-label classification methods are of paramount importance as they are more and more required in applications such as news categorization, music categorization [7], medical diagnosis, etc. In the traditional single-label classification, the objective is to learn and predict a single label from the set of labels $\mathbf{L}$ for each instance. If the set of labels $\mathbf{L}$ only consists of two elements, it's a "binary classification" problem and if it has more than two elements, it is named to be "Multi-class classification". In "Multi-label classification", there are instances that are assigned to a subset of labels $\mathbf{S} \subseteq \mathbf{L}$.

As stated in [15], there exists several natural **problem transformation** methods that can be used to tackle multi-label classification problem by turning the original problem to one or more single label problems. One way is to randomly choose only one label from the set of labels an instance has and disregard all the other labels for that instance. This way, the

problem turns into the traditional single-label classification. Another way to transform the problem is to just discard all multi-label instances existing in the dataset. However, these two ways have obvious drawbacks as they throw away a lot of information. A third way known as **Label combination** method is to consider each set of labels as a new label, in other words create the power set $\mathbf{P(L)}$ of labels and consider each of them as a new label. This way, all the combinations of labels are considered as a new label. This will result again in transforming the problem into single-label classification. The good thing about this method is that it doesn't assume labels to be independent. However, this method results in us having too many classes and few instances available for each class and also suffers from time complexity. In [3], the authors have proposed an Ensemble Method called RAKEL that picks a random subset of a certain size from Labels and then uses the label combination method just discussed. We also experimented RAKEL on our dataset, but the results were not satisfying so we did not further investigate them.

The most common problem transformation is called **Binary Relevance** method which we are going to discuss in III-A. There are other approaches that have been suggested for problem transformation too which we are not going to discuss.

Apart from problem transformation, it is possible to adapt and modify single-label algorithms to be useful in context of multi-label classification. In [15] several of these algorithms such as AdaBoost[11], Multi-label K-nearest neighbor (ML-KNN)[8], One-Vs-All Support Vector Machines (SVM), Neural Network, etc. have been briefly explained.

In what follows we are going to briefly explain the models we used for our multi-labeled problem of predicting genres. Other models can be used in future work to compare the results.

A. Binary Relevance

It can be argued that the most common problem transformation method to transform a multi-labeled problem to single-labeled problems is the Binary Relevance (BM) method. It takes advantage of low computational complexity. BM learns $|\mathbf{L}|$ binary classifiers. What it actually does is that in the original dataset, if an example contains a label l, marks it as l and else marks it as $-l$. This is done for each of the labels $l \in \mathbf{L}$. One obvious drawback of BM is that it doesn't capture the dependencies and relationships between labels; for instance $\{$Drama, Romance$\}$ have strong dependency and it's very likely that a movie has such label (which is actually true and has a high probability after doing experiments). On the other hand $\{$War, Comedy$\}$ are not likely to be seen as a label together for a movie. To implement BM, we used MOLearn[2] which is Methods for Multiple-Output Learning in python and is adapted from **MEKA** framework [16]. Also used one-vs-rest approach available in sklearn package.

B. Classifier Chains

Classifier Chains (**CC**) were introduced in [10] by Read et al. as a multi-output classification method. Classifier Chains are actually a very smart twist to Binary Relevance which results in the model taking advantage of low memory just like BM but at the same time overcoming the drawback of label-independence that was present in the BM. So classifier chains just like BM, create $|\mathbf{L}|$ binary classifiers but what is different is that they include previous predictions as feature attributes. So as stated in the original paper, a chain $\{\mathbf{C_1}, ..., \mathbf{C_{|L|}}\}$ is formed. Each $\mathbf{C_i}$ in this chain, learns and predicts the binary association of label l_i given the feature space extended with the $\{0, 1\}$ label association of all previous predictions in the chain $l_1, ..., l_{i-1}$.

An extension of **CC** which we also used is Monte-Carlo search for Classifier Chains (**MCC**) [4]. **MCC** technique searches the label-path space and tries to find a good chain sequence since it's obvious that the order of chain affects accuracy. As stated in the paper, it takes much longer than (**CC**) to be run and this was the case in our experiment as well. For implementing both **CC** and **MCC**, we took advantage of MOLearn. We used two different classifiers (SVM, logistic regression) on both methods as internal single-label classifier to compare the results.

C. Decision Tree

Decicion Trees have been adapted and modified in such way to be used for Multi-label classification. Decision trees are a supervised learning classifier that do not have parameters so they don't need much tuning and therefore simple to use. From the features they tend to learn some simple decision rules and based on these rules, they create a model that can predict the label of an instance. In [2] several approaches to the induction of decision trees for Multi-label classification have been suggested. The Decision tree classifier in sklearn package can handle multi-label settings so we used it to experiment on our dataset. Also in the paper [11] two extensions of AdaBoost algorithm named Adaboost.MH and Adaboost.MR have been proposed. They use weak classifiers to concentrate on labels that will be most beneficial to finding an accurate ranking of labels and classification rule. We did not implement these algorithms and they can be investigated in future work.

D. Neural Network

Neural Networks are able to capture and model label dependencies in the output layer hence several architectures for Neural networks in the context of multi-label classification have been proposed. In [9], authors claim to have proposed the first multi-label neural network algorithm in literature named **BP-MLL**. This neural network architecture has traditional fully connected layers. What differs however, is that the classical back propagation algorithm for neural network has been adapted by replacing its error function with a new function. This new error function is designed such that the labels belonging to an instance are to be ranked higher than those not belonging to that instance. Consider an instance x_i of the data set. Let E_i be the error the neural network is making on that point. Let $c_j^i = c_j(x_i)$ be the actual output of network for x_i on the j-th label. Y_i is the true label set for x_i and $\bar{Y}_i$ is

[2]https://github.com/jmread/molearn

TABLE II: Summary of experimental results for different methods on 60,000 instances of **IMDB** data with document representation of Word2Vec

	F1-score(Weighted)	Hamming_loss	Hamming_score	Exact_match
Logistic regression-BR	0.448750	0.068087	0.931912	0.204646
Logistic regression-CC	0.454515	0.065223	0.934776	0.219797
Logistic regression-MCC	0.460195	**0.065061**	**0.934938**	0.228888
SVM-BR	0.447457	0.068488	0.931511	0.217494
SVM-CC	**0.484497**	0.069300	0.930699	**0.263757**
SVM-MCC	0.451098	0.067388	0.932611	0.253858
DecisionTreeClassifier	0.294696	0.111922	0.888077	0.122020
BPMLL	0.407025	0.110804	0.889195	0.095050

TABLE III: Summary of experimental results for different methods on **IMDB** data with document representation of Word2Vec keeping only Top 10 most frequent labels. On 60,000 instances of data.

	F1-score(Weighted)	Hamming_loss	Hamming_score	Exact_match
Logistic regression-BR	0.676207	0.099831	0.900168	0.571717
Logistic regression-CC	0.698297	0.104124	0.895875	0.641414
Logistic regression-MCC	0.703751	0.101262	0.898737	**0.647979**
SVM-BR	0.692063	0.103872	0.896127	0.569191
SVM-CC	0.697978	0.104966	0.895033	0.637373
SVM-MCC	**0.713951**	**0.089724**	**0.910275**	0.643939
DecisionTreeClassifier	0.352362	0.109870	0.890129	0.15575
BPMLL	0.383465	0.100889	0.899110	0.190909

TABLE IV: Summary of experimental results for different methods on 60,000 instances of **IMDB** data with document representation of Doc2Vec

	F1-score(Weighted)	Hamming_loss	Hamming_score	Exact_match
Logistic regression-BR	0.271919	**0.107564**	**0.892435**	0.122626
Logistic regression-CC	0.274786	0.132492	0.867507	0.121111
Logistic regression-MCC	0.282648	0.145137	0.854862	0.138585
SVM-BR	0.347411	0.143033	0.856967	0.133333
SVM-CC	**0.353616**	0.115311	0.884689	**0.153221**
SVM-MCC	NN	NN	NN	NN
DecisionTreeClassifier	0.210024	0.118109	0.881890	0.140808
BPMLL	0.271764	0.123033	0.876966	0.094040

TABLE V: Summary of experimental results for different methods on 28,000 instances of **IMDB** data with document representation of Tf-idf.

	F1-score	Hamming_loss	Hamming_score	Exact_match
Logistic regression-BR	0.205600	0.183926	0.870734	0.107400
Logistic regression-CC	0.220000	0.182050	0.81795	0.113572
Logistic regression-MCC	**0.314600**	0.139220	0.86078	**0.123572**
DecisionTreeClassifier	0.262341	0.111428	0.888571	0.031230
BPMLL	0.240000	**0.096071**	**0.903928**	0.120000

the is the complementary set of Y_i. **BP-MLL**'s error function has been defined as:

$$\mathbf{E}_{BPMLL} = \sum_i \mathbf{E_i} = \sum_i \frac{1}{|Y_i||\bar{Y}_i|} \sum_{(k,l)\in Y_i \times \bar{Y}_i} \mathbf{exp}(-(c_k^i) - c_l^i) \tag{2}$$

Thus it can be seen that Y_i which are labels belonging to the instance x_i, should be ranked higher than those not in Y_i and hence the correlation between labels is considered in this error function. We used three hidden layers and the input layer has 300 neurons which is the vector size for plot representation derived from Word2Vec and Doc2Vec, and the output layer has 28 neuron each corresponding to one of the possible genres. In the context of neural networks for multi-labeled data, a simple NN approach with recently proposed learning techniques for large-scale multi-label text classification tasks has been proposed in [5] which can also be investigated in future work.

IV. Evaluation Metrics

In this section before presenting the results of algorithms, we will briefly introduce the evaluation metrics that we used for the multi-label classification. In the setting of multi-label classification, measures of accuracy and performance evaluation are different from those of classical single-label classification (accuracy, precision, recall, F1 score) because they should be able to capture the notion of being partially correct. We used three different evaluation measures in the scope of this project.

A. F1 Score

F1 measure is actually the harmonic mean of Precision and Recall. Micro-averaged F-measure, calculates evaluation metric globally by counting the total true positives, false negatives and false positives and is mainly used in the setting of single-label classification. Macro-averaged F-measure

calculates evaluation metric for each label, and finds their unweighted mean. Assume p_i and r_i are the precision and recall for the i-th label respectively. Then the macro-averaged F-measure which is used in multi-label setting is given by:

$$F1score_{macroaveraged} = \frac{1}{|\mathbf{L}|}\sum_{i=1}^{|\mathbf{L}|}\frac{2 \times p_i \times r_i}{p_i + r_i} \tag{3}$$

Weighted-averaged F-measure calculates evaluation metric for each label, and finds their average, weighted by support (the number of true instances for each label). This modifies macro-averaged F-measure to deal with label imbalance. We used Weighted-averaged F-measure as an evaluation metric.

B. Hamming Loss

Hamming loss is one of the most frequently used criterion in literature. It gives the percentage of wrong labels to the total number of labels. The lower the hamming loss, the better the performance of method we are investigating. Assume in our multi-label dataset $\mathbf{y_i}$ is the ground truth label for instance i and $y\prime_i$ the prediction and that $\mathbf{n}$ is the number of samples. Hamming loss is given as follows:

$$HammingLoss = \frac{1}{n}\sum_{i=1}^{n}\frac{xor(y_i, y\prime_i)}{|\mathbf{L}|} \tag{4}$$

C. Exact Match

Exact match ignores the notion of being partially correct for the multi-label setting and considers those labels that have been partially correctly predicted as incorrect. $\mathbf{I}$ is the indicator function

$$ExactMatch = \frac{1}{n}\sum_{1}^{n}\mathbf{I}\{y_i = y\prime_i\} \tag{5}$$

Exact match is a harsh measure since it doesn't distinguish the two different cases of being partially correct and being completely wrong.

V. Experimental Results

We have experimented different methods discussed in III on our dataset and the results of each method has been evaluated against the metrics defined in IV. Although not considered big data, the **IMDB** dataset we did our experiments on was much larger than a personal computer resources' capacity. It was even not plausible to run the algorithms on all the data using a Google Virtual Machine. Hence after getting the document-representation for plots, we couldn't manage to work on all the data especially when using Bag-of-Words (explained in II-A); we only managed to implement algorithms on 28,000 instances of plot summaries, since by then we had already exceeded 51000 features for vectorizing the plots. The values for $\mathbf{BR}$, $\mathbf{CC}$, and $\mathbf{MCC}$ with SVM as classifier are not given here as it was not scalable to suit the size of the dataset.

For Word2Vec (explained in II-B), we used two approaches to get the results. First, we experimented the methods in III on about 60,000 instances of data with all combinations of labels being available. Then, we decided to only choose the top 10 most frequent labels, for instance top 1 {drama}, top2 {documentary}, top3 {drama and romance}, top 4 {comedy, short}. By keeping only those instances that have labels belonging to top 10 most frequent labels, 128,000 instances of data (almost half of the original data set) are kept. From this set, we randomly chose 60,000 instances to evaluate the different aforementioned methods for multi-label classification. We expect this setting to give a better performance as the number of possible labels has been reduced and we also managed to run it on more data.

For Doc2Vec (explained in II-C), we have run experiments on about 60,000 instances of data just like the setting for Word2Vec. To recap, for Word2Vec and Doc2Vec document representations we used 60,000 instances of data to evaluate different algorithms. In addition, we considered another setting for Word2Vec which only keeps instances that have top 10 labels (about 128,000 instances of data) out of which we randomly chose 60,000 instances to conduct experiments on. For tf-idf representation, we evaluated different methods on only 28,000 instances.

The evaluation of each classifier is conducted by splitting the training set by the ratio of 67-33% as training and validating sets, where the training set is fed to a classifier and the validating set is used in evaluation phase after which a score is assigned to each classifier. We tried to tune the hyper parameters of the internal classifiers of **MCC** and **CC** (logistic regression and SVM) using grid search. Tuning the parameters of SVM with rbf kernel really improved the results (using C=30, gamma= 1.15). In tables II, III, IV, V the predictive performance of all explained algorithms have been displayed for document representation of plot summaries achieved using Word2Vec, Word2Vec-top 10 most frequent labels, Doc2Vec and Bag-of-Words respectively.

As it was expected, experimental results for methods using Word2Vec and keeping only instance whose labels are among the top 10 most frequent labels, show very good results (table III). It can be seen that **MCC** with SVM classifier is giving the high F1-score of 0.71. We can also see an exact match of 0.64 with **MCC** method with logistic regression classifier. It means that %64 of labels were correctly predicted (no even partially correct but %100 correct) and this is mainly because we have reduced the possible number of labels and only kept 10 most frequent labels. One nice thing to see in this setting, is plotting low dimensional representation of average Word2Vec embedding vectors for top 10 most frequent labels. This has been visualized in figure 2 using UMAP. Uniform Manifold Approximation and Projection ($\mathbf{UMAP}$) for Dimension Reduction has been proposed in this [6] very recent paper. It has the advantage of being faster over t-SNE and it's claimed that it can preserve more of the global structure in comparison with other methods. In figure 2 it can be seen that the the word embeddings for the labels have been very nicely separated into different clusters. This can imply that getting the average of Word2Vec embedding vectors is meaningful and has yield a good representation.

We used Decision Tree Classifier as a baseline because they run quite fast but as it can be seen their result is not very much

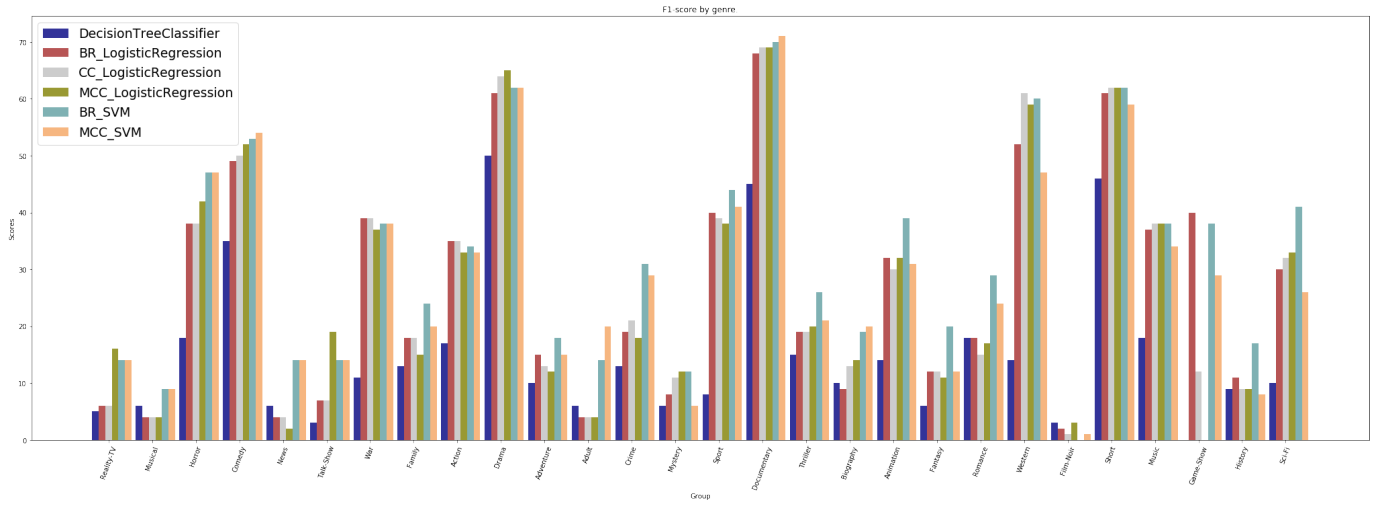

Fig. 3: F1-score per genre Word2Vec on **IMDB** dataset

able to compete against other methods we have taken into account. **BPMLL**'s results is an improvement over Decision Tree Classifier. Tuning hyper parameters of the neural network and choosing a better architecture might improve this model.

One very nice thing to see is that with all classifier chains method are performing best in all settings. As it was mentioned in theory and as expected, they are being an improvement over binary Relevance because they are not ignoring the dependencies that are available between genres. The best result for f1-score is achieved with **CC** having an internal SVM classifier with rbf kernel and it's equal to 0.48. We assume this result to be a good in comparison to works done in literature [17] (although not exactly the same **IMDB** data set); given that we have not been able to perform methods on all the data. The results of all methods and all metrics are somewhat disappointing with Bag-of-Words since it is failing to give a meaningful representation for plots. It is not able to get the intuition behind each plot summary and convey it through its poor representation. It must also be taken into account that it is being investigated on fewer instances of data.

Since the best results was achieved with Word2Vec representation of plots, in figure 3 we have plotted F1-score by genre of each model. It can be seen that decision tree classifier is being biased towards the most popular genres (Drama, Documentary, Short, and Comedy) and is under-performing on less popular genres. Classifier chains are almost giving the highest F1-score per genre, however, it can be noticed that binary relevance is competing closely or even better on some genres.

Hence, in the end we observed that classifier chain with Word2Vec representation gave best results since they are capturing the meanings behind the plots and dependencies between labels better than other methods we investigated. A lot of improvement can be done on the different settings for instance, using more data can definitely augment Doc2Vec representation.

VI. Conclusion

In this paper, we explored several Machine Learning methods to predict movie genres based on plot summaries. First, we investigated three different methods to get vectorized representation of plot summaries. Then we introduced different methods for classifying multi-labeled data since genre classification had to be done on the multi-labeled dataset of **IMDB**. Comparing the results, we came to the conclusion that Classifier chain methods of **MCC** and **CC** performed best evaluated on metrics we have previously defined (Using Word2Vec to represent plots as vectors). Other methods such as Adaboost.MH, Adaboost.MR, MLKNN, etc. can be investigated in this setting to compare the results. It would also be nice to run the algorithms on all the data set when having more computational resources.

References

[1] Google pre-trained word2vec. 2013.

[2] S. D. H. B. Celine Vens, Leander Schietgat. Decision trees for hierarchical multi-label classification. 2008.

[3] D. L. Jesse Read1a, Luca Martinoa. Efficient monte carlo optimization for multi-label classifier chains. 2013.

[4] D. L. Jesse Read1a, Luca Martinoa. Efficient monte carlo optimization for multi-label classifier chains. 2013.

[5] E. L. M. I. G. J. F. Jinseok Nam, Jungi Kim. Large-scale multi-label text classification revisiting neural networks. *Springer, Berlin, Heidelberg*, 2014.

[6] J. H. Leland McInnes. Multi-label collective classification. 2011.

[7] T. . O. Li. Music Genre Classification with Taxonomy. *IEEE International Conference*, 2005.

[8] Z.-H. Z. Min-Ling Zhang. A k-nearest neighbor based algorithm for multi-label classification. *IEEE international conference on Granular Computing*, 2005.

[9] Z.-H. Z. Min-Ling Zhang. Multi-label neural networks with applications to functional genomics and text categorization. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 2005.

[10] H. e. a. Read, Pfahringer. Classifier chains for multi-label classification. *Springer US*, 2011.

[11] Y. S. ROBERT E. SCHAPIRE. A boosting-based system for text categorization. *Machine Learning, 39, 135168, 2000*, 2000.

[12] M. d. R. Tom Kenter, Alexey Borisov. Siamese cbow: Optimizing word embeddings for sentence representations.

[13] I. S. Tomas Mikolov and G. C. Kai Chen, Jeffrey Dean. Distributed representations of words and phrases and their compositionality. 30, Apr. 2013.

[14] Q. V. L. Tomas Mikolov. Distributed representations of sentences and documents. 30, Apr. 2014.
[15] G. Tsoumakas and I. Katakis. Multi label classification: An overview. *International Journal of Data Warehousing and Mining*, 2007.
[16] G. Tsoumakas and I. Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. 2016.
[17] P. S. Y. Xiangnan Kong, Xiaoxiao Shi. Umap: Uniform manifold approximation and projection for dimension reduction. *SIAM International Conference on Data Mining*, February 13.