Text Classifier with Custom Feature Extraction for Genre Identification

Aishwarya Jauhari
Matrikel-Nr.: 226084

Amar Shivaram Pallassana Gopalakrishnan
Matrikel-Nr.: 226015

Baalakrishnan Aiyer Manikandan
Matrikel-Nr.: 225818

Manish Bhandari Vipinraj Bhandari
Matrikel-Nr.: 226011

Manjusha Vishvanath Pattadkal
Matrikel-Nr.: 229763

I. Introduction

Recent developments in Natural Language Processing emphasize on the fact that it is necessary for a human reader to understand the several levels of meaning and abstraction in order to truly understand a document. Different aspects of the text can be helpful in classification, as well as several other purposes, which are becoming significant subjects of focus as text-based applications are becoming more diverse with a large amount of information and sources. The category of "Text" doesn't just consist of short online articles or reviews of products and services but encompasses much more than that. Literature is just one of the few other sources of texts which is of interests after recent strides in the area of Digital Humanities. Literature is an art that conveys even the complicated themes by the means of long narratives.It's classification through the use of underlying information such as themes, plot-lines, characters and setting is the need of the hour.

Text classification is usually built in a broad range of contexts for classifying texts or tweets or even organizing much larger documents such as publications and books. Well-known examples of text classification include sentiment analysis, topic labeling, language detection, and many more.Classification of text genre can be beneficial to several text-based applications. For example, if the genre of every document is prior assigned, the information retrieval results can be more efficiently presented to the user according to their preference.Several approaches to provide a solution to the genre identification problem would help with information retrieval and recommendation systems as well. It helps in the study of diverse literature, themes, and genre. Genre identification also contributes to the full-length literature understanding and comprehension with complex themes throughout an entire narrative. One major challenge which might occur is the high dimensional feature space and closely related features of more than one genre. Along with this, one peculiarity of the subset of the corpus is the high-class imbalance which also greatly influences the model. In our project, we build a classifier that uses genre-specific custom features to address these issues.

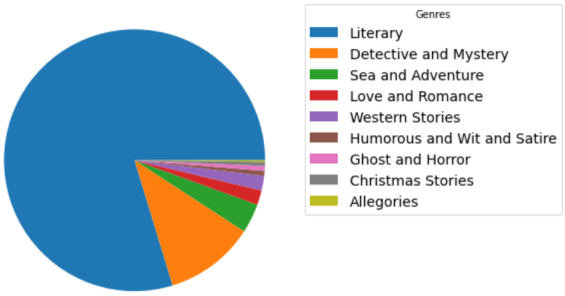

Fig. 1. A pie plot representing the Class distribution of the data set based on their *Genres* . The distribution shows high imbalance in the data set with majority of the documents belonging to the class *Literary*

II. Dataset

Project Gutenberg is a library of over 60,000 books written by over 142 authors on various genres. For our project, we consider a subset of 996 books. Out of these 996 books, 2 books were found to be empty which were omitted from the final data set. The final data set consists of 994 books spanning over 9 different genres. These genres include *Literary, Detective and Mystery, Sea and Adventure, Love and Romance, Western Stories, Humorous and wit and satire, Ghost and Horror, Christmas stories and Allegories*. The distribution of these books based on their genre is shown in figure 1. The plot shows that majority of the books belong to the genre *Literary* followed by *Detective and Mystery*. The rest of the genre constitute only a small portion of the data set. Further, since the documents are books and most of the words don't occur in every document, the vocabulary built on the data set is very large resulting in lengthy but sparse document vectors.

III. Methodology

Most of the traditional classification approaches for textual data represent data in the form of vectors which are further fed to the classification model to obtain results. These vectors are called feature vectors and are unique for individual documents. Each dimension of the vector corresponds to a separate term. If a term occurs in the document, its value in the vector is non-zero. There are several different ways in which these values, also known as (term) weights can be computed. The

definition of term depends on the application. Typically terms are single words, keywords, or longer phrases. If words are chosen to be the terms, the dimensionality of the vector is the number of words in the vocabulary (the number of distinct words occurring in the corpus [1]. This can be problematic in case of a huge corpus with lengthy documents. The resultant vector for a document may have millions of dimensions. There are several feature extraction techniques available such as filtering methods, fusion methods, clustering methods, and deep learning approaches which can be used to reduce the dimensionality of the vector. We combine [2] these traditional filtering methods with our methodology to extract custom features that will form the basis for the feature vectors. We extract these custom features using two different techniques. The resulting feature sets from both the techniques are then combined to form the final feature vectors.

A. Custom Feature Extraction from list of Common words

TF-IDF is one of the most popular term-weighting schemes used today. TF-IDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a corpus. It is often used as a term weight in the feature vectors. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general [3] . This property of tf-idf can be exploited to identify words which are unique to a document but do not appear in other documents. This idea can be extended to class level as well. If all documents of a particular class are augmented to create a single document then for a total of 9 class, we'd have 9 documents. Now, if we apply tf-idf weighting to these documents and sort the vectors of these documents with the highest tf-idf value first, then we can essentially pick out top k words with highest tf-idf values which are unique to this class. However augmenting all the documents of a class together to create one single document can create other problems, one of them being producing very lengthy vectors. However, this idea can be simplified and still be applied at the document level. First we create 9 lists. Each of these lists represents a class(genre) from our corpus. Then, for each document in the corpus, we generate their respective tf-idf vectors. Each of the vectors are then sorted in decreasing order with highest value of weights first and the top 1000 values from each vectors are selected. Each of these values corresponds to a word. We add these words to the respective list representing the class of the document. For example, say we have a document from class *Literary*. We find the tf-idf vector of this document, sort it and then select the top thousand values. The corresponding words are added to the list representing class *Literary*. We do this for all the documents in our corpus. The resulting lists contain top 1000 words from each document belonging to the particular class which this list represents. We now extract custom features for each document from this list. This is done by comparing each document with each of the 9 lists and counting how many

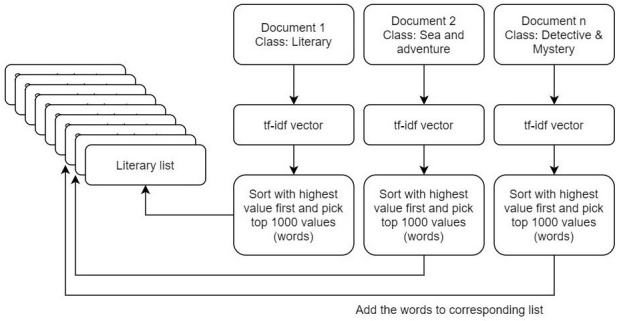

Fig. 2. Creating a list for each class by taking top 1000 tf-idf weighted words from each document of respective class

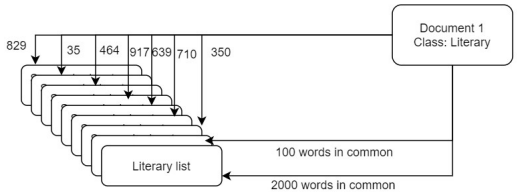

Fig. 3. Comparing a document with each of the 9 lists to get the feature vector for the document. The length of the feature vector is 9 where each entry represents the number of common words in a document with the corresponding list, normalised with the length of the list.

words from each lists are present in the document. The simple intuition is that a document belonging to the class literary will have most of the words from the *Literary* list and lesser words from other lists. This is similar to taking the unnormalized similarities between the document and each of the list. We do not need to normalize the count by the length of the document since we just consider if a word from the list is present or absent in the document and do not take the number of times this word appears in the document. However, it is important to normalise this count with the length of the list since each of the lists have different lengths. This is because the lists contains top 1000 words from each of the documents of a particular class. Say if class *Literary* has 100 documents then the list *Literary* will have 100*1000 = 100000 words but if a class *Detective and Mystery* has 10 documents then the list *Detective and Mystery* will have just 10*1000 = 10000 words. Hence it is important to normalise the count with the length of the lists. Following this procedure for every documents, we will have 9 values for each document representing the normalised count from each lists and we consider these as our feature vectors.

B. Custom Feature Extraction from Word Emotion

A book is best described from the emotion it portrays. For example a book belonging to *Love and Romance* genre is expected to contain high amount of words expressing Love, care and related emotions and less number of words expressing other emotions. Similarly, a *Detective and Mystery* book is expected to have high amount of words expressing emotions

such as Violence, crime, and other negative emotions. These different categories of emotions can capture the genre of each book. The idea is that each genre can be represented by certain sets of emotions and if a count of number of words expressing each emotion is taken, then a book can be described by the number of words of each category of emotion it contains. We use these different categories of emotions as our features and the count of number of words of a particular category of emotion in a document as respective feature value. We use *Empath* [4] to identify different emotions present in a document. Empath is used to analyze text and identify different lexical categories. There are over 200 built-in categories. Other categories can also be built according to one's need. For our project which is restricted to 9 genres, we select 48 categories that in combination best represent different genres. The selected categories are *Hate, cheerfulness, aggression, envy, crime, attractive, masculine, prison, health, pride, dispute, nervousness, suffering, ridicule, computer, optimism, divine, sexual, fear, childish, religion, internet, surprise, worship, medieval, death, healing, politics, celebration, violence, love, ancient, urban, science, youth, fun, white_collar_job, technology, philosophy, monster, negative_emotion, positive_emotion, weapon and feminine* [5]. For each document in the corpus, we count how many words of each category it contains. We use this count as the feature value. It is intuitive that the value of the domain is greater for the categories which represent that particular genre.

In the end, we combine the features produced from both the methods. Hence, each document is represented as a vector of length 57 (9 features extracted with the list of common words and 48 features extracted from emotions a document portrays.) The same is done for all the documents in the training set and is then fed to train a classification model.

IV. Implementation

The process flow is depicted in figure 4. Each of them are further explained below.

A. Pre-Processing

Data is extracted from HTML files containing HTML tags which are not informative and hence are removed. Pre-processing takes place in two phases. First is tokenization and stopword removal, and second is removing words with single letter [6].

Tokenization is the process where the text is broken into tokens which words, numbers, punctuation, etc. The NLTK Word tokenizer is used to tokenize each document on the basis of white spaces and punctuation marks. We remove unnecessary punctuation and numerical tokens. If the created tokens are characters then they are stored. Therefore, for HTML tags like $< p >$ after tokenization only p is passed further. Stopword removal is done by dictionary based approach. NLTK has predefined dictionary for the English stopwords according to which all the stopwords are eliminated from our tokenized corpus [7]. Single letter words including the pre- processed HTML tags are also removed in our final pre-processing step.

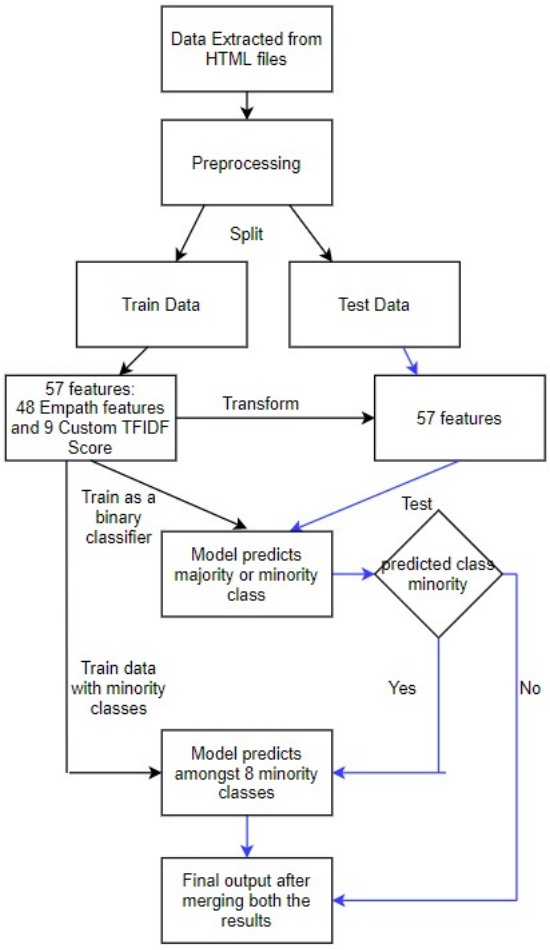

Fig. 4. Process flow

B. Model Building

Pre-processing is the first step of implementation. Once pre-processing is done, data is cleaned and is ready to be analyzed further. Before going onto feature representation, data is split into train set and test set. 25 percent of the data is kept to test the model.

Due to the high class imbalance, usual traditional classification methods fail to predict all the classes and the prediction of the model always is seen to be of the majority class. To overcome this we use the one v/s all approach. In this approach, we train two classifiers. One for the majority and the minority class (all the classes apart from the majority (*Literary*) are considered together considered as one class) and other for just the minority class. We implement this by first modelling on the whole data, 8 minority classes are given label as 1 and majority class 'Literary' is given label as 0. This is a binary classifier that predicts if the class is majority or minority. The second model is only trained to predict the minority class. With a test document, it is first fit

TABLE I
EVALUATION RESULTS

No.of Samples	No. of Features	Method	Classifier	Train Recall	Test Recall	F1 Score	Cohen Kappa Score
994	56	Normal	SVM	79.19	61.04	0.6484	0.3177
994	56	Normal	RF	97.45	84.33	0.7843	0.3632
115	56	Normal	SVM	93.02	41.37	0.3556	0.2472
115	56	Normal	RF	98.87	31.03	0.2329	0.1761
994	56	Binary	SVM	84.69	79.51	0.8032	0.4255
994	56	Binary	RF	98.52	82.73	0.7855	0.2744
202	56	Minority	SVM	85.43	62.74	0.6746	0.4859
202	56	Minority	RF	100	72.55	0.6882	0.5405

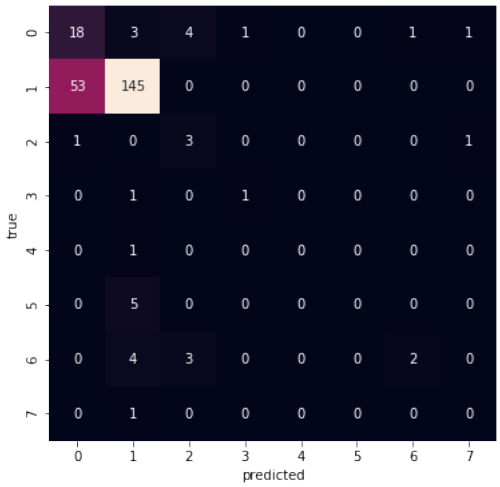

Fig. 5. Confusion Matrix of SVM

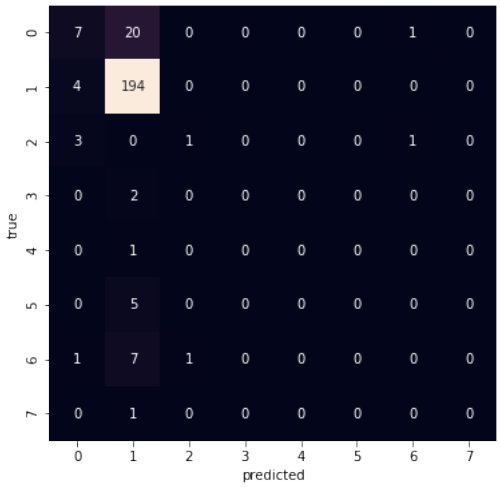

Fig. 6. Confusion Matrix of RF

through the binary classfier to predict whether the document belongs to the majority or the minority class. If the document is found to be of the majority class, then the prediction process stops there and we get an output label as *Literary* else if the binary model predicts it as a minority class, then the second classifier predicts which minority class does the document belong to and outputs the corresponding class. This significantly improves the classification performance as it is evident from the confusion matrix in 6 and 5.

V. EVALUATION

The data set is split into train, validation, and test sets to get a good estimate of generalized model performance. The training data is used to train the model and validation data is used to select the best model hyper-parameters. The independent holdout test data is then used to obtain generalized model performance. We have considered several models to test for different scenarios. These different scenarios are depicted in the table I. We selected Random Forest (RF) and Support Vector Machines(SVM) as our choice of classification algorithm. These models are proven to perform best for text classification tasks. There are several variations of parameters with which these model were tested. First, whether the model is trained and tested using the entire available data or the data is undersampled. The Second variation pertains to the type of learning algorithm used i.e. either SVM or RF. The last variation is the way these algorithms are implemented to build a classifier on the data namely ,the traditional method of training a single classifier and the one v/s all classifier. Since a large amount of imbalance is present in the data (in the case when the data is not undersampled), we train a binary classifier which just trains to predict whether the document belongs to the majority or the minority class. Then on top of this, a minority classifier is trained, which predicts label of the minority class the document belongs to if the one VS all classifier had predicted it as the minority class. However, when working with the undersampled data, we just train the traditional classifier. The data is undersampled to mitigate the class imbalance issue. We follow a heuristic of randomly sampling 20 documents from each class. If any class has less than 20 instances, then all the instances from that class are considered. Thereby a total of 115 instances were picked in this custom undersampled data. The model was trained on both SVM and RF classifiers after appropriate model hyper-parameter selection. The evaluation measures chosen are weighted Recall, weighted F1-Score, and Cohen Kappa Score. The results obtained are depicted in table I. After obtaining classifiers of Binary and Minority with respect to SVM and RF, we combine them and by doing so we get the following F1 Scores. The F1 Score obtained for SVM was 0.7747 and that

obtained for RF was 0.7755. The confusion matrices obtained are visualized. Based on the obtained matrices we can infer that the model built using the SVM algorithm is better in predicting the minority classes. Hence we select SVM as our base model for this dataset. The problem of imbalanced dataset is addressed and tried to overcome using the binary classifier approach. Thereby using this new novel approach, we are able to get a good classification performance using the SVM algorithm.

VI. Conclusion

The 19th Century English Literature Gutenberg corpus containing approximately 996 books (including the books which were empty) were processed using a standard pre- processing pipeline. Emotional features were extracted from each of these books using the EMPATH library enabling us to identify the underlying theme. Words that specifically characterised a genre fiction were identified and extracted as feature for classification using tf-idf as weighting scheme. The specific problem of Class Imbalance was handled using a binary classifier and then followed by a Multi-label Classifier which enabled the training process to accommodate the smaller classes making it more efficient and accurate. A trained SVM classifier gave us good classification results in spite of the skewed class distribution.

The classification process with the feature extracted and engineered from the fictional text rather than standard bag of words approach, has enabled us to obtain drastically reduce feature set size/dimensions and at the same time obtain similar accuracy. These features are more explainable to a naive user of the classifier whose understanding of the genre is based on the underlying theme or plot of a work of fiction. Further progress can be made in certain directions. One is to consider different techniques while extracting lists for each individual classes. Mutual information measure indicates the relationship between a word and the class label. This can be used instead of tf-idf weighting to extract the lists.

References

[1] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[2] S. Polley, S. Ghosh, M. Thiel, and A. Nurnberger, "SIMFIC: An Explainable Book Search Companion," p. 4, 2019.

[3] C. Fautsch and J. Savoy, "Adapting the tf idf vector-space model to domain specific information retrieval," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, (New York, NY, USA), p. 1708–1712, Association for Computing Machinery, 2010.

[4] E. Fast, B. Chen, and M. S. Bernstein, "Empath: Understanding topic signals in large-scale text," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 4647–4657, 2016.

[5] E. Fast, B. Chen, and M. S. Bernstein, "Empath: Understanding topic signals in large-scale text," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, (New York, NY, USA), p. 4647–4657, Association for Computing Machinery, 2016.

[6] A. I. Kadhim, Y. Cheah, and N. H. Ahamed, "Text document preprocessing and dimension reduction techniques for text document clustering," in *2014 4th International Conference on Artificial Intelligence with Applications in Engineering and Technology*, pp. 69–73, 2014.

[7] E. Loper and S. Bird, "Nltk: The natural language toolkit," 2002.