ATiML Project: Genre Identification on a subset of Gutenberg Corpus

Himanshi Bajaj
Faculty of Computer Science
Otto von Guericke University
Magdeburg, Germany
himanshi.bajaj@st.ovgu.de

Nandish Bandi
Faculty of Computer Science
Otto von Guericke University
Magdeburg, Germany
nandish.bandi@st.ovgu.de

Steve Simon
Faculty of Computer Science
Otto von Guericke University
Magdeburg, Germany
steve.simon@st.ovgu.de

Sujith Nyarakkad Sudhakaran
Faculty of Computer Science
Otto von Guericke University
Magdeburg, Germany
sujith.nyarakkad@st.ovgu.de

***Abstract*—This project is about Genre Identification on 996 books which are taken from Gutenberg Corpus. Corpus contains 9 genres. We are classifying each fiction book with a genre label not by using a bag of words model. First, features are extracted from each book using some popular libraries available for example, nltk, spacy etc and we have used three models and with the help of grid search, model selection is performed. Later test data is used to check how models are performing on unseen data and evaluation and visualization of results is performed.**

Keywords - Feature Extraction, Class Imbalances, Supervised Learning, Fiction

I. Motivation and Problem Statement

We are trying to solve challenges which are faced while considering a bag of words model for modelling text data. This technique is unique in a way that instead of having a vocabulary extracted from the corpus and creating vectors, we are focusing on designing meaningful features and using them further for the classification task. A bag of words model maps textual data to vectors with real values, sentences or the entire document will contain 1 in the vector for the word being present and zero for word not there in vocab. But, in this project relevant features are related to fiction books, some of the features are sentiment analysis, ease of readability etc.

The intent of this project is to represent each data point (document) in the form of several relevant features by making use of some predefined libraries. Each document is a book with hundred of thousands of words so feature extraction plays an important role in a way that the entire document can be represented in the form of meaningful real-values. Dealing with such a big corpus requires a lot of memory space, handling dataset in a way such that memory issues will not be there is a necessary requirement. Also, dataset has a class imbalance problem, as one class takes up 80% of the corpus, so model selection and evaluation should be done keeping in mind this problem.

II. Dataset

Project Gutenberg is a library that contains more than 60,000 e-books in which most books are very old literary works. We are provided with a subset of Project Gutenberg which contains 996 books. These 996 books are a collection of fiction work by single authors. Along with HTML files of all the books, there is metadata provided like: book_id, genre (class label), and author name.

A. Data Structuring:

Documents are labelled with one of the 9 labels. 9 Genres are: Literary, Detective and Mystery, Sea and Adventures, Western Stories, Love and Romance, Ghost and Horror, Humorous and Wit and Satire, Christmas Stories and Allegories.
Number of Documents in the dataset: 996
Number of labels: 9
Class Label with the highest number of books: Literary
Class Label with the lowest number of books: Allegories

B. Data Insights:

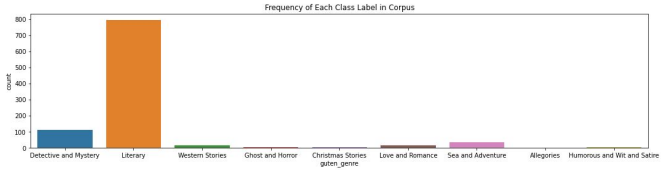

Fig. 1. Count Plot to show frequency of different class labels

Figure 1 is not able to clearly depict to us how many books belong to which category , pie chart i.e. figure 2 is giving better insights in terms of percentage.

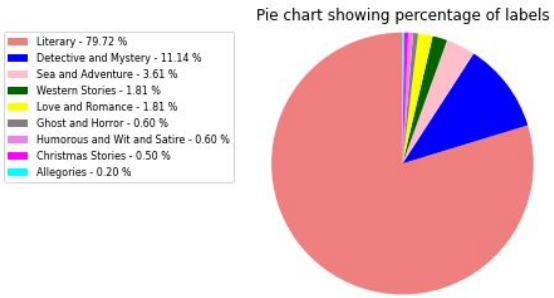

Fig. 2. Pie chart to show percentages of class labels

From figure 2, it could be noticed that literary comprises of almost 80% of the corpus, and the rest of the documents vary from 11 to 0.2% which shows that dataset has class imbalance problem.

III. Concept

A. Data Preprocessing

After reading csv file to a dataframe, we first took the book id and replaced .epub with .html, so that it was easier to read files from the directory. With the help of labelencoder from sklearn.preprocessing, we converted target labels to values between range 0 to number of classes-1. Since html files contain paragraph tags, we removed them by making use of re library and store it as a string. We considered the length of the text to be 3050000 to save the processing time.

Most of the content in the books has unwanted information such as punctuation, stop-words etc. We started by removing the punctuation and replacing them with white spaces so that we only have words in our text. We followed up by separating the text into individual tokens using nltk.tokenize and selecting only the alphabetic characters. We removed the most frequently occurring words by importing the nltk English stop words. But these pre processing steps are done only for some features, some of the features took the whole book as an input to avoid any loss of information.

B. Feature Representation and Extraction

We have in total 7 features and we have taken inspiration from [1] , for all of our features.

[]

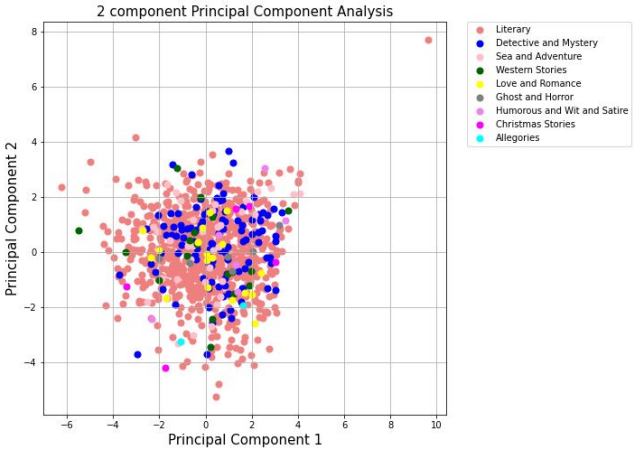

Fig. 3. PCA Analysis of our feature set reduced to 2 components

In figure 3, we have done PCA on the features extracted, first standardization was performed by making use of sklearn.preprocesing.StandardScalar and using sklearn.decomposition.PCA features are reduced into two dimensions.

1) Plot Complexity: Often the high fantasy complex Stories have too many characters in the novel. Therefore, we made an assumption that the difficulty of the plot would increase with an increase in the number of characters. The text we received after removing the HTML tags is used to extract the number of characters in the book. We extracted the POS tags for all the sentences using nltk.postag. The NLTK tagging function gives tokens and part of speech tag in the form of a tuple, e.g. ('William', 'NNP'). We took all of the Singular Proper Nouns to get the names of the characters. We've had a few problems here, firstly giving some VERBS as proper nouns and secondly, giving the first and last names as two different proper nouns. For example, "William" and "Jarker" were considered different words for the count. We have solved this problem by considering the two proper nouns as one, if they occur together in the text. We used the collections. Counter to count the number of times it appeared. We set 10 as the minimum number of times it should appear in the count to be considered as a character in the novel. Nevertheless, this was still giving some random words as characters in the book. We were thinking to create some custom words to eliminate those, but we thought it was too much to do, given that we have close to 1000 books.

In the meantime, we've come across library spacy that performs Named Entity Recognition. The nlp function generates the entities for all the sentences in the text. The named entities group the words into different categories, such as person, place, etc. We picked all the words that have PERSON as an entity label and found them to be characters in our novel

2) Male and Female Pronouns: Male and Female pronouns are two features which are being calculated simultaneously in a function defined as *gender roles*. Gender roles can help us see how many pronouns related to male characters and female characters are present in the book. If male pronouns are larger in number as compared to female pronouns, novel could be have male protagonist or if female pronouns are more in number, lead character is female.

In this function *gender roles* we are passing the whole data frame and making use of two for loops for the computations. Firstly we have created a pronoun list with 'he' and 'she' in it and two empty arrays to store the count of pronouns. In order to calculate pronouns, we have first taken each record, converted it to a bigram and stored it as a Counter dictionary containing count for each bigram appearing how many numbers of times in the record. These bigrams are appended into the original dataframe for further computations. First for loop is used to access bigrams of each datapoint and with that second for loop takes the first word of bigram and checks if it is not he or she it is deleting all those bigrams, and checking in remaining bigrams if the first word is he or she and increasing the count by 1, appending to empty lists and finally inserting them into the dataframe.

Figure 4 is a pair plot for columns he pronoun and she pronoun with hue as a class label. Pair plot is taken from

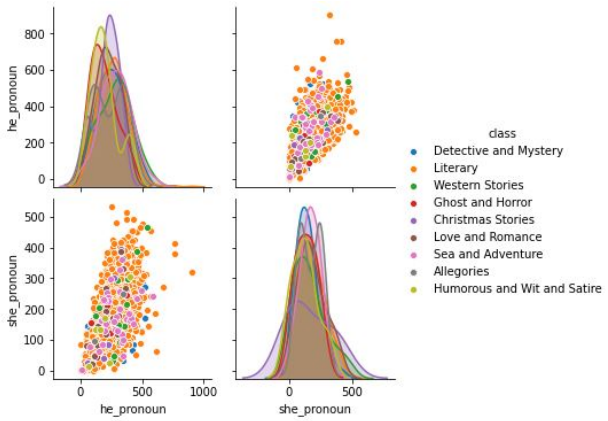

Fig. 4. Pair Plot for He and She Pronouns with hue as class label

visualization library seaborn.

3) Ease Of Readability: Ease of Readability can be described as the ease with which a reader is able to understand the content of written material. How much a person will be able to understand a text will be associated with Readability score. Values of this score can tell us whether the text is easy or difficult to understand. There could be various methods to check ease of readability for example: Flesch-Kincaid, Raygor Readability Graph, Dale Chall Readability Score, and Flesch Reading Ease. We are calculating the Flesch Reading Score.

Flesch reading score is calculated with the help of average sentence length and average word length in syllables. Formula is given as: 206.835 - (1.015 x ASL) - (84.6 x ASW). Average Sentence length is nothing but a number of words divided by a number of sentences and Average word length in syllables is defined as a number of syllables divided by number of words. In order to calculate these values we have defined 8 functions.Function *sentence break* takes the list string i.e whole text of book and breaks it into separate sentences. We have made use of spaCy to perform this operation. We are calling nlp object on our book and storing it in Doc. This doc will contain all the processed data, but we need only sentences, we are returning doc.sents from this function. One good thing about using spaCy is we don't lose any information from the entire text. Function *word count* is taking all the sentences returned by sentence break and calculating the count of words. Function *sentence count* is counting the total number of sentences which are contained in each book. Function *avg sentence length* calls word count and sentence count and checks for the condition if there are no sentences it will return zero. While calculating these values we came across two files which contained no text. Later we removed these two files from the file generated and dealt with 994 records instead of 996. Function *syllables count* returns syllables in each word of a sentence for all the sentences. Textstat is a python API developed to calculate different statistics related to a text. We are making use of its function syllable count. This textstat function uses pyphen python package in order to perform syllable calculations. Now that we have total syllables in the book, we can calculate the average syllables per word. In order to do that we have a function *avg syllables per word* , this function gets total syllables and total words in the book and check for the condition if there are no words in the book it will return zero, in order to avoid zero division error, otherwise, it will calculate average syllable per word by dividing individual total values. We are making use of a function legacy round which is defined in textstat, which is calculating a round value of the result by making use of functions floor and copysign of math library. Lastly, function *flesch reading score* is putting ASL and ASW values in the formula and returning the score. If the score lies in the range of 90 to 100 that shows book is easy to understand by middle school students. If, score lies in the range of 60 to 70 , it is easily understood by students belonging to the 13-16 age group. If the score lies between 10 to 30, the age group of university graduates can understand the text.

4) Lexical Diversity: "Lexical diversity can be described as the range and the variety of vocabulary used in a text by the writer or speaker" [2]. Lexical diversity indicates the level of 'lexical richness' and is given by the ratio of different unique word stems (types) to the total number of words or tokens. Several measures have been used to evaluate the lexical Diversity including Type-Token Ratio (TTR), Moving average TTR (MATTR), Measure of lexical textual diversity (MTLD), Hypergeometric Distribution D (HDD) etc. Oldest Lexical diversity metrics such as TTR and its transformations are heavily dependent on text length and text samples containing a large number of tokens give lower values for TTR as the writer might re-use several function words.[3]

However, more complex transformations and sophisticated formulae such as MTLD, HDD are argued to be independent of text size. In our approach, we have used the Hypergeomteric Distribution D (HDD) measure [2] which is based on hypergeometric distribution. For each lexical type in a given text, probability of finding any of its tokens is calculated from a random sample of 42 words which is taken from the same text. The sum of all such calculated probabilities for all lexical types gives the HDD final score. Lexical diversity is greater as the Type-Token ratio approaches the value of 1 indicating a high degree of lexical variation

5) Sentiment Analysis: Sentiment Analysis (Opinion Mining) is a technique that takes the emotional context from sentences and categorize into different degrees. The emotions of the text along with other features can be used for developing effective text classification models. There are various python based libraries for performing Sentiment Analysis. This model uses python NLTK library for performing Sentiment

Analysis. NLTK provides VADER(Valence Aware Dictionary and Sentiment Reasoner) Sentiment Analysis tool[4] captures sentiments and quantifies it based on the emotions involved. The idea is that these kinds of emotions can give certain insights on genre of the books. The VADER lexicon is a tool which is trained with very diverse text that includes tweets, amazon reviews, movie reviews, editorial snippet etc. The Vader lexicon that has been used also deals with intensifiers. Intensifiers or degree modifiers impact the statement by varying the degree of positiveness/negativeness in the sentence.
E.g. Sentences such as 1. Car is extremely good 2. Car is extremely good! 3. Car is good 4. Car is moderately good
As shown in the sentences above, it can differentiate and quantify the degree of positiveness within these sentences which varies only with the degrees and punctuation. Each and every book is given with positive, negative, neutral and compound scores. The compound score is normalized between -1(most extreme negative) to 1(most extreme positive). This is useful only when sentiment detection has to be performed. But in this case, since the major goal is not to detect the sentiment but to obtain some information from sentiment to classify the genre of book, the features such as positive, negative and neutral scores are being used.

6) Sentence Length: We have considered one low-level feature Sentence length. We are taking each record and calculating total number of sentences present in that data point.

IV. Implementation

Data is available in the form of HTML files and we are provided with book id, so we accessed files by making use of book id's and took book id's in a separate array to use it for looping over the whole dataset. We took each HTML file's content, preprocessed it, extracted features and inserted them into a new dataframe with all other features. Later we just stored that data frame as a csv file.

We initially ran the whole dataset in our system, but due to less RAM available, it was taking a lot of time and it failed after a few attempts. We then split the dataset in 6 parts of files ranging between 46 to 200 and ran it on Google Colab. On an average, it took roughly 240-250 minutes for some 200 files. Later, after getting features for all parts, we clubbed them. We noticed while executing zero division error for two files - pg38685 and pg34164, so after we had features for all 996 files, we removed these two data points. And we had a final csv file with no null values for further computations.

For modelling, we have considered various models from scikit learn library.

A. Models under Consideration

1) Multinomial Naive Bayes: Naive Bayes classifier is well known machine learning classifier which has vast range of application in NLP and other classification problems. Although it is simple, it is said that in some cases it outperforms some of the decision tree classifiers.

The math behind the model is quite simple to understand as it works on Bayes theorem which relies on maximum likelihood probability estimation. It finds the most probable class given a document. Multinomial naive bayes implements naive bayes for multi nomially distributed data. The algorithm provided by sklearn takes three hyper parameters such as alpha, fit_prior, class_prior. The hyper parameters fit_prior, class_prior are associated with prior probabilities. As prior probabilities are equally likely to occur, these parameters are set with default values. The other parameter alpha which is Laplace smoothening parameter is tried with a different value and the one that gives maximum performance metrics is selected by performing cross validation using grid search.

2) Support Vector Machine: We know that SVM is one of the best ML algorithms for supervised learning, and is also widely used for classification problems. We used the Support Vector Classification which is a subclass of the SVM (Support Vector Machine) class.
First, we learned the hyper parameters for SVC from Grid Search, which tests all the combinations we have defined in the grid parameters. We've got the optimum parameters from best estimator attribute and we've set the same parameters. We used the sklearn package to perform SVC. We used RBF Kernel to transform the input space to another dimensional space. We kept the misclassification factor as 1 to control the trade-off for decision boundary and we have set the Gamma value to be 0.001.Next, we trained the Linear SVC on the training data and then predicted the target using the test data. The corresponding results are mentioned in the section V.

3) Random Forest Classifier: Random Forest Classifier as the name implies uses a large number of individual decision trees which operate as an ensemble(use multiple learning algorithms for better predictive performance). Random forest model makes use of two important concepts namely: Training data points are randomly sampled when trees are build (bootstrapping) and when splitting nodes we consider random subset of features.

We have used random forest implementation from the *sklearn.ensemble* module utilizing the *RandomForestClassifier* class. The parameters for the model include: *n_estimators* which is the number of trees in the forest, *bootstrap* which decides if the whole sample is used to build the tree or whether bootstrap samples are being used, *random_state* which controls the randomness of the bootstrapping of the samples. We have experimented with multiple values for n_estimators and other parameters to have an understanding of how the model is being affected and the evaluation is presented in the section V.

B. Dealing with Imbalanced Data

Data set is said to be imbalanced if there are classes having unequal distribution. For example in our dataset, literary class has 794 records out of 996 total. So, this is a highly imbalanced dataset. There are many techniques to deal with imbalanced data for example : Upsampling, Downsampling, generating synthetic data, using ensemble algorithms technique. Upsampling and Downsampling further have many techniques apart from just doing it randomly. These techniques might help us in getting better classification results as well as balancing all the classes in the dataset.

In order to deal with the imbalance problem, there were few packages that needed to be installed. Sklearn by default in Google Colab has version 0.22 which is not supported by package imblearn which is used for performing sampling methods. So, we first installed 0.24 version of sklearn and then installed imblearn package.

1) Up Sampling: Up Sampling or Over Sampling is a method to deal with imbalanced dataset, where we increase records of the minority classes. There are various ways of up sampling dataset like: Randomly, SMOTE, Cluster based over sampling, MSMOTE. Technique which we are using for our dataset is SMOTE.

SMOTE stands for Synthetic Minority Over-Sampling Technique. In [5], authors are up sampling all the classes present in minority by generating synthetic data. Authors have said to generate new data by exploring attribute space instead of data specific region. As per [5], synthetic data is being created by taking k nearest neighbours of the minority classes and joining them with the help of a line segment and neighbours are decided randomly.

For our dataset, we have taken original dataframe and used object *SMOTE* with k nearest neighbours as 1 and then created synthetic data for all the minority classes. and created a balanced dataframe out of it, split the data into train and test applied standard scaling on it and, used Train from the balanced dataframe , but for testing we have taken original dataframe's test split. We have tried two models on this new data *SVM* and *XGBoost* and then recorded all evaluation measures.

2) Under Sampling: Down Sampling or under Sampling is a method in which we try to remove few data points from the majority class. There could me various methods of doing so, for example: Random undersampling, Tomek Links , cluster centroids based etc. We have made use of Tomek Links under sampling technique.

Tomek links are pairs of data points that belong to different classes but are very close. This algorithm starts searching for these pairs and then remove majority class out of it.

In our case, we have picked the original dataset of features and used object *TomekLinks* from the library *imblearn under sampling* to first create tomek links and then use that to sample our original data. Later, we have split the data into train and test, standardized it and tried models *SVM* and *XGBoost* on down sampled data. Almost 70 records were removed from majority class Literary. We have then calculated evaluation metrics on the final data.

V. Evaluation

We have made use of famous evaluation metrics like F1-Score, Precision, Recall and Accuracy. Accuracy is not relevant here, because of imbalanced data, we are comparing our models on rest of the three. We have considered macro and weighted averages in this case.

1) Macro: In this metrics are calculated for all the class labels present and further unweighted mean is computed. Class Imbalance in not taken into consideration.

2) Weighted: Metrics are calculated for all the class labels and weighted average is determined with the help of support. Support is how many true records are present for each class label. In this, class imbalance is taken into consideration.

Precision: Precision tells us how much our model is accurate in a way that instances that are predicted to be positive, out of them are actually positive.

Recall: Recall which is also known as sensitivity, calculates how many of true positives are actually predicted as true positives.

F1-Score: is derived from both precision and recall. It performs better than Accuracy in case of class imbalance. It is nothing but harmonic mean of Precision and Recall.

The given data set is tested with the following models and the results are documented as follows:

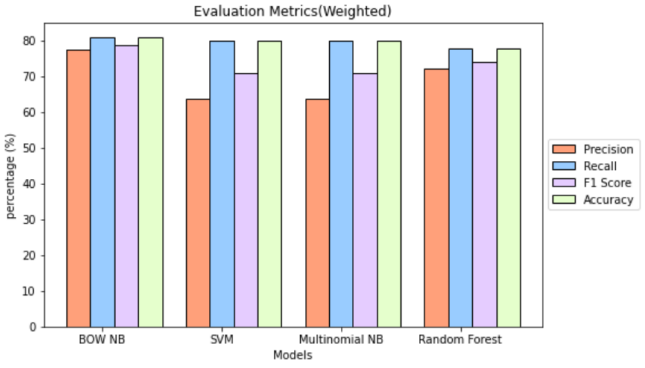

Fig. 5. Model performance with average as weighted

TABLE I
PERFORMANCE OF ORIGINAL FEATURE DATA SET WITH AVERAGE AS MACRO

Classifier[b]	Evaluation Metrics (Macro)(%)			
	Precision	Recall	F1	Accuracy[c]
BOW NB	24.4	26.8	25.2	80.9
SVM	9.9	11.1	12.5	79.9
NB	9.9	12.5	11.1	79.8
RandomForest	19.0	15.8	16.5	77.8

TABLE II
PERFORMANCE OF ORIGINAL FEATURE DATA SET WITH AVERAGE AS WEIGHTED

Classifier[b]	Evaluation Metrics (Weighted)(%)			
	Precision	Recall	F1	Accuracy[c]
BOW NB	77.6	80.9	78.8	80.9
SVM	63.8	79.8	70.9	79.9
NB	63.8	79.8	70.9	79.8
RandomForest	72.0	77.8	73.9	77.8

The performance of the models is evaluated using metrics such as Precision, Recall, F1 Score and Accuracy as shown in the table above. Table 1 depicts macro scores of all the metrics for different classifiers. In macro score, each classes are weighed equally and these metrics are computed.Table 2 depicts weighted scores of all the metrics for different classifiers. These scores are obtained by assigning weight to the classes and then computing these metrics.

We have also used gridsearch in sklearn for parameter estimation. The hyper parameters associated with models are tuned by applying different values for them and cross validating against the given dataset. Finally, we have fixed a hyper parameter which gives better results.

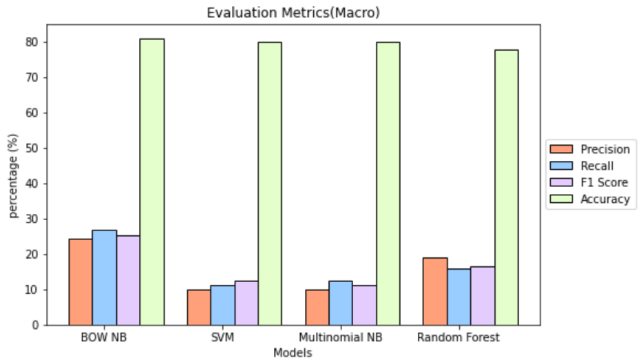

Fig. 6. Model performance with average as macro

Fig.5 and Fig.6 gives the overall visual representation of the performance metrics applied on different classifiers. As shown, these metrics are computed for weighted and non weighted classes.

Table 3 and Table 4 gives us results of how over sampled

TABLE III
PERFORMANCE OF BALANCED DATA SET WITH AVERAGE AS MACRO

Dataset[a]	Classifier[b]	Evaluation Metrics (Macro)(%)			
		Precision	Recall	F1	Accuracy[c]
OverSampled	SVM	35.3	80.7	42.4	56.2
OverSampled	XGBoost	25.2	44.6	28.5	56.2
UnderSampled	SVM	9.7	12.5	10.9	78.3
UnderSampled	XGBoost	15.8	14.9	14.8	78.9

TABLE IV
PERFORMANCE OF BALANCED DATA SET WITH AVERAGE AS WEIGHTED

Dataset[a]	Classifier[b]	Evaluation Metrics (Weighted)(%)			
		Precision	Recall	F1	Accuracy[c]
OverSampled	SVM	81.1	56.2	61.5	56.2
OverSampled	XGBoost	79.3	56.2	61.8	56.2
UnderSampled	SVM	61.4	78.3	68.8	78.3
UnderSampled	XGBoost	68.9	78.9	72.8	78.9

and under sampled dataset respectively performs for two models SVM and XGBoost considering two averages - Macro and Weighted.

VI. CONCLUSION

Through this paper we have tried to identify the Genre of a book by extracting features from corpus and using these features as input to a classifier model. The task of assigning a literary genre to a book from a corpus requires thematic, syntactic and semantic analysis and we have tried to capture some of these aspects while developing our classification system. Features we focused tried to explain different elements of fiction such as Plot complexity, Gender Roles, Ease of readability along with lexical richness and sentiment analysis to understand the theme around which the whole book revolves. We also explored techniques to mitigate class imbalance such under sampling and up sampling due to the highly imbalanced nature of data we dealt with. Along with genre identification different machine learning approaches such as MultiNomial Naive Bayes, Support Vector Machines, Random Forest Classifier were presented and evaluated as solutions for assigning genre.

As future work, we can extend our study by using larger data sets and different set of genre types. Additionally topic modelling based features such as Latent Dirichlet Analysis can be explored to enrich the feature set. We have used relatively simple machine learning models for classification and significant amount of work can be done using Convolution Neural Networks (CNNs) to achieve performance improvements.

In order to reduce class imbalance we can make use of Deep Learning technique Machine Translation. Pass each book from set of encoders and decoders and change language of a book from English to any other language suppose French

and then translate it back to English again, Content of the book will not be same entirely but we can get new records which are related to previous ones. This way we can balance our data and further extracts features out of it.

REFERENCES

[1] S. Polley, M. Thiel, M. Kotzyba, and N. Andreas, "SIMFIC : An Explainable Book Search Companion."

[2] P. M. McCarthy and S. Jarvis, "vocd: A theoretical and empirical evaluation," *Language Testing*, vol. 24, no. 4, pp. 459–488, 2007.

[3] M. Ströbel, E. Kerz, D. Wiechmann, and Y. Qiao, "Text genre classification based on linguistic complexity contours using a recurrent neural network," *CEUR Workshop Proceedings*, vol. 2134, no. Cm, pp. 56–63, 2018.

[4] C. Hutto and E. Gilbert, "A parsimonious rule-based model for sentiment analysis of social media text," 2014.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE : Synthetic Minority Over-sampling Technique," vol. 16, pp. 321–357, 2002.