News Article Classification - Project Report

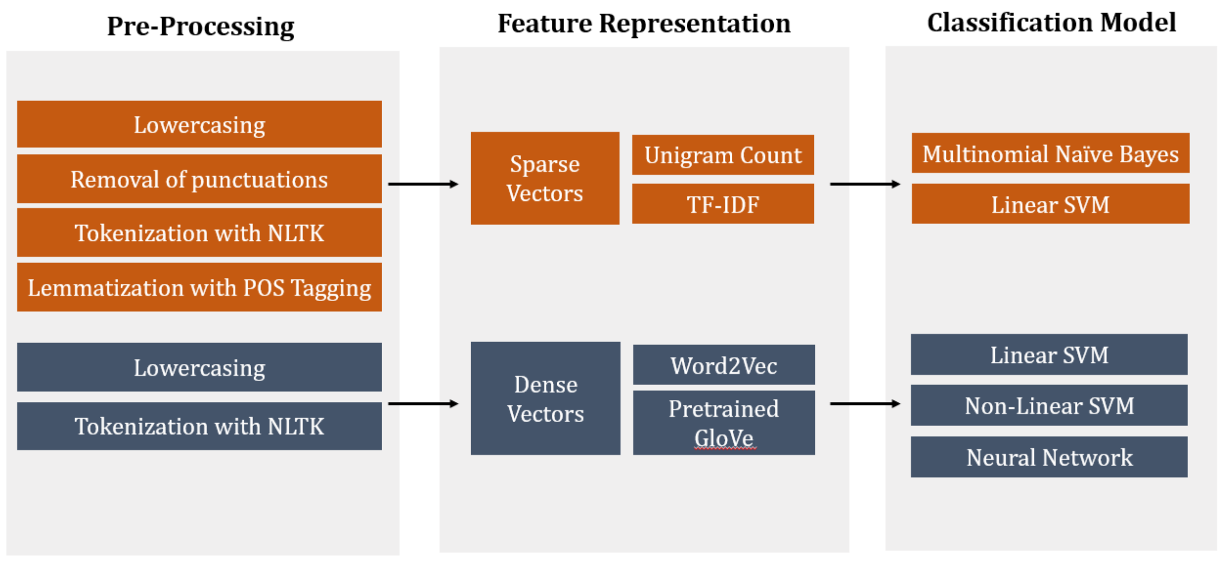
Shruti Ravichandran (sr67), Yashas Prashanth (yashasp2)

Introduction

The aim of this research project is to identify the best performing text processing and modeling pipeline for categorizing news articles using the headline and a short description of the article. The dataset which is used for this investigation is obtained from Kaggle. It contains headlines and short descriptions of news articles from the Huffington Post published between the years 2012 to 2022. This dataset is in JSON format with labeled categories like 'Politics', 'Sports', and 'Arts' which can be used for supervised learning. For example, one of the rows is:

```
{
        "link":"https://www.huffpost.com/entry/hulu-reboot-should-you-watch-it_n_6324a099
        e4b0eac9f4e18b46",
        "headline": "'Reboot' Is A Clever And Not Too Navel-Gazey Look Inside TV Reboots",
        "category": "CULTURE & ARTS",
        "short_description": "Starring Keegan-Michael Key, Judy Greer and Johnny Knoxville, the
        Hulu show follows the revival of a fictional early 2000s sitcom.",
        "authors": "Marina Fang and Candice Frederick",
        "date": "2022-09-20"
}
```

This study explores the use of sparse and dense feature representations, the requisite text preprocessing for each, and various combinations of supervised classification techniques to find the modeling pipeline that produces the highest evaluation metrics in this multi-class text classification task. Various pre-processing techniques, like lowercasing, tokenization, lemmatization, and POS Tagging are applied according to what is best suited for the downstream pipelines.



Descriptive Statistics

The Huffington Post dataset consists of ~209K news articles from 2012 to 2022, with the articles categorized into 42 unique categories such as Politics, Sports, Travel, etc. Upon checking the dataset for nulls and blank strings, 6 articles with no heading and ~19k articles with no descriptions are found.

The distribution of data across years is plotted and evaluated to understand how recent the data is. 91% of the data is distributed between 2012 to 2017, indicating that the data is fairly recent, but not the most up to date. However, given that the data is recent enough to reflect modern English usage, it is deemed to be fit for the task of text classification for this project.

The distribution of data across 42 categories is also plotted and evaluated to understand if there is an inherent imbalance of samples across the prediction classes. This is indeed found to be true with the top 7 classes making up 51% of the samples. It can also be observed that some minority classes like ‘STYLE’, ‘TASTE’, and ‘COLLEGE’ that are quite similar in meaning to other classes like ‘STYLE & BEAUTY’, ‘FOOD & DRINK’, and ‘EDUCATION’ respectively. There are also some categories like ‘WORLDPOST’ and ‘THE WORLDPOST’, and 'CULTURE & ARTS', 'ARTS & CULTURE', and 'ARTS’ that seem to be the exact same classes with minor differences in the category name. These categories are noted to be possible options for combining classes as a way of combating class imbalance and reducing noise should the need arise based on preliminary evaluation.

Pre-Processing

The dataset is in json format which is read and stored into a dataframe. The ‘headline’ column containing the article headlines, ‘category’ column containing the class labels and ‘short description’ column containing a brief description of each article are identified to be important for this analysis. The 6 records with no headlines are removed from analysis as no features can be generated for these, leaving behind 209,521 articles forming the input for the classification pipelines.

Different sets of pre-processing techniques are employed for sparse and dense vectors. For representation of features with sparse vectors, every headline and short description is pre-processed by converting the text to lowercase, getting rid of extra spaces and replacing all non-alphanumeric characters with a single space. Lemmatization is not performed for the initial part of the analysis, but added later based on preliminary results (discussed more in detail later). For a dense vector representation, the text is converted to lowercase and tokenized, but punctuations are kept as is to retain as much contextual information as possible for training the word embedding model. For both sparse and dense vector pipelines, the text is tokenized using NLTK’s word_tokenize. Finally, the training text is obtained by combining the contents of the preprocessed headline and short description columns.

The dataset is also split into a training set and test set using scikit-learn’s ‘train_test_split’, using a 80-20 split. The training set is used to train the corresponding models and the performance of the models are evaluated on the test set. The stratify parameter is set to maintain a similar distribution of classes in both training and test sets so that the model performance can be evaluated in similar imbalance conditions.

Feature Representation and Modeling (using Sparse Vectors):

Features to represent the text are constructed using counts of the words, by considering unigrams - as the length of the headline and short description is small, averaging around 32 words. The 'CountVectorizer' of scikit-learn is used to construct count-based unigram features on the training dataset. Considering all the words produce a very large number of features (87025), the max_features parameter is set to 20,000 to only select the 20,000 most frequent features. The top 20000 features were also selected using Mutual Information Gain, but this resulted in a similar model performance as compared to frequency based feature selection, hence all results reported with sparse representation have frequency based feature selection. The features generated on the training dataset are used to vectorize the test dataset. Stratified k-fold cross validation strategy is used on the training vector to train all the downstream classification models as an initial step to ensure that the models are generalizing well. Following that, each model is trained again on the training set and evaluated on the test set. First, a Multinomial Naive Bayes classifier model is trained and various performance evaluation metrics like accuracy score, and weighted precision, recall, and F1 score are calculated. A Naive Bayes model is trained on the training split, and evaluated on the test split of the dataset. Focusing on the F1-score, this pipeline yields a score of 56% on the test data.

Next, features to represent the text are constructed using TF-IDF, as the frequency of occurrences of various terms and across documents is closely related to the overall theme of the text - the category of the news article. A unigram TF-IDF vectorizer is fit on the training dataset to generate features, which are also calculated for the test dataset. The F1-score is found to be 41.6% when trained on the entire dataset and evaluated on the test dataset. It is noticed that tf-idf feature representation has much poorer performance compared to count based feature representation. Hence, a Linear Support Vector Machine is trained using unigram count vectors with a regularization parameter of 1.0 on the training count vector. Before training the SVM, the data is scaled using standard scaler, and the target variable undergoes label encoding. This pipeline yields a F1 score of 51.6% on the test data.

Feature Representation and Modeling (using Dense Vectors):

Representation of features using dense vectors is first performed using the skipgram version of word2vec. The preprocessed and combined column with headline and short description is used to train the models to get semantically encoded word embeddings with a vector size of 100. The embeddings trained using the train split of the dataset result in ~108K word embeddings. Any additional words in the test set that aren't present in the train set are ignored to get the word embeddings for the test dataset. For both the train and test set, the document embedding is calculated by averaging out the word embeddings across each dimension. This becomes the feature set for training the classification models.

The feature set with dense vectors is used to train text classification models including NN based and non-NN based models. Among non-NN based models, linear & non-linear SVM and Logistic Regression are chosen as both these models are known to work well with high-dimensional data and can handle dense vectors efficiently. The StandardScaler is used to standardize the feature vectors because both SVM and Logistic Regression can benefit from feature scaling. The class_weight parameter is set to ‘balanced’ for all three models so that classes are weighted inversely proportional to their frequency to offset for the high class imbalance in the dataset. The classification threshold is left at a default value of 0.5. The SVM model with RBF kernel is found to yield the best performance with an F1-score of 54%, followed by linear SVM with an F1-score of 49% and Logistic Regression with an F1-score of 48%.

Finally, a simple neural net model with one hidden layer was trained on the training dataset with a batch size of 64 and a validation split of 20% for 20 epochs. The learning rate is set to 0.001 and categorical cross-entropy is used as the loss function. The training accuracy of the model converges at 54% and validation accuracy at 56%. The accuracy of the model on the test set is 55**%** and F1-score is 51%.

The following table summarizes the results of the various pipelines explored so far:

Pre-Processing	**Feature Representation**	**Feature Selection**	**Model**	**F1-Score**
Lowercasing, Removal of Punctuation and Tokenization	Unigram Count	Most Frequent (20000)	Naive Bayes	56%
	Unigram Count	Most Frequent (20000)	Linear SVM	52%
	TF-IDF	Most Frequent (20000)	Naive Bayes	42%
Lowercasing and Tokenization	Word2Vec (skipgram)	-	Linear SVM	49%
	Word2Vec (skipgram)	-	Non-linear SVM	54%
	Word2Vec (skipgram)	-	Logistic Regression	48%
	Word2Vec (skipgram)	-	NN with 1 hidden layer	51%

Intermediate Analysis and Improvements:

The results obtained with the above approaches are analyzed to come up with hypotheses on possible issues and infer clear next steps to attempt improving the best performing models from those tried above.

Lemmatization with POS Tagging for Sparse Vectors
In sparse vector representation, it is noticed that the unigram count-based features result in a very large feature space leading to highly sparse vectors. While feature selection could mitigate this to an extent, the fact that it simply drops a lot of words remains which could be leading to loss of crucial information. Therefore, in order to reduce the feature space more meaningfully while conserving important contextual information, POS tagging with lemmatization is chosen. In this method, first NLTK's pos tagging is used to assign a POS to each word in a text, and then the POS tagged words are lemmatized.

Preserving more contextual information in word embeddings for Dense Vectors
In the dense vector representation with word2vec, the obvious hypothesis is that the embeddings trained on the current dataset may not capture the semantic meaning of words properly due to limited context coming from the dataset. To test if this is indeed true, pretrained GloVe embeddings with 1B words are used. These embeddings have been trained on a larger set of data and hence could have better context.

Addressing class imbalance
When looking at the f1-score at class level, it can be observed all the models show a much better performance for majority classes like 'Style & Beauty' (with F1-score of 71% in case of linear SVM with word2vec embeddings) while performing much poorer for minority classes like 'Latino Voices' (with F1-score of 8% for linear SVM with word2vec embeddings). Further, it is noted that this difference increases with decrease in overall model performance. Thus, improving the class imbalance was another aspect that is sought to be improved. A couple of approaches were explored to address this issue. First, the classes that are identified to be similar in meaning (discussed in the Descriptive Statistics section) are combined to result in reduction of classes from 42 to 36 classes. This leads to less noise and more samples in the remaining classes. Second, two different sampling strategies are used to balance the samples in the training dataset. The first strategy is to oversampling the minority classes using SMOTE. The second is a hybrid strategy where the top 3 classes are randomly undersampled and the bottom 32 classes are oversampled using SMOTE to 4th class' n-count. The second sampling approach resulted in better performance on the test dataset as it has less number of synthetic samples and hence is found to result in more generalized models.

Model performance after Improvements

With sparse vectors, Naive Bayes is obtained as the best performing model with an F1-score of 58% with unigram count vectors as features after preprocessing using lemmatization with POS tagging, combined classes, and hybrid sampling. It is notable that while the TF-IDF model falls a little short with an F1-score of 57%, it has shown a significant improvement in performance after introducing the model improvement measures.

With dense vectors, using pre-trained GloVe vectors did not lead to any significant improvement in any of the models, indicating that quality of word embeddings may not be the issue here. However, measures to address data imbalance did result in 2-3% gain in model performance for Non-linear SVM as well as NN models. The non-linear SVM model is observed to perform the best when the data is preprocessed to combine classes and sampled using the hybrid strategy. The NN model performed the best on combined classes without sampling as it started to overfit despite strong regularization when trained on sampled data. The best performing model for dense vectors is found to be non-linear SVM with 56% F1-score.

Feature Representation	Feature Selection	Model	Test F1-Score (before)	Test F1-score (after)
Unigram Count	Most Frequent (20000)	Naïve Bayes	56%	58%
TF-IDF	Most Frequent (20000)	Naive Bayes	42%	57%
Word2Vec	-	Non-linear SVM	54%	56%
		Neural Network	51% (one hidden layer)	53% (two hidden layers)

Insights & Conclusion:

Looking at the table above, it can be observed that text classification pipelines with sparse vector representations have performed marginally better as compared to dense vector representations. However, the difference is not significant enough to be able to conclude that sparse vectors are better than dense vectors for this study. Unigram count vectorization gave better results as compared to TF-IDF, but the performance became comparable once the imbalance in the dataset was addressed, suggesting that TF-IDF is more sensitive to imbalance.

In terms of the various models experimented with, Naive Bayes and Non-Linear SVM are the best performers for this classification problem. While the neural network based models are not too far behind, it is observed that these models suffered from overfitting despite of introducing regularization measures when the dataset was sampled to counter the data imbalance. Given that simple NN models with one or two hidden layers failed to generalize well, it is not likely that more complex models like CNN or RNN would do so either.

This study delves into two broad ways to improve the overall performance: enhancing the feature set and addressing the imbalance between prediction classes. Upon tackling each of these one by one, it is observed that addressing the imbalance using various strategies yielded a better increase in performance as compared to enhancing the features for both sparse and dense vector based pipelines. While vectorization and classification models do play an important role in the model training, this indicates that a high degree of class imbalance is a key factor driving down the overall model performance, and should be carefully dealt with especially while working with a large number of classes.

As a future scope, perhaps it would be worthwhile to train BERT embeddings on this dataset to see if these conclusions still hold true. Training an ensemble model is another unexplored option that can be considered to further test these observations.

References:

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., ... Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Retrieved from https://tensorflow.org

Camacho-Collados, J. (2018, August 23). On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. arXiv. https://arxiv.org/abs/1707.01780

Dhar, A., Mukherjee, H., Dash, N.S., et al. (2021). Text categorization: Past and present. Artificial Intelligence Review, 54, 3007–3054. https://doi.org/10.1007/s10462-020-09919-1

imbalanced-learn. (n.d.). imblearn.over_sampling.SMOTE. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

Loper & Bird (2002). NLTK: The Natural Language Toolkit. arXiv preprint arXiv:cs/0205028.

Misra, R. (2022, September 23). News category dataset. arXiv. https://arxiv.org/abs/2209.11429

Misra, R., & Grover, J. (2021). Sculpting Data for ML: The first act of Machine Learning. ISBN 9798585463570

Pandas development team. (2023, December 8). User guide. Retrieved from https://pandas.pydata.org/docs/user_guide/index.html#user-guide

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

Pennington, J., Socher, R., & Manning, C. D. (2014, August). GloVe: Global vectors for word representation. Retrieved from https://nlp.stanford.edu/projects/glove/

Řehůřek, R. (2022, December 21). Word2Vec Tutorial. Gensim. Retrieved from https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html#