Project Name

Developing a Multi-Label Classification Model Using the RCV1 Dataset for Accurate News Articles Classification

Student Name: *Utkarsha Kshirsagar.*

RIT Email ID: uk9263@rit.edu

Course: DSCI 633: Foundations of Data Science
Instructor: Dr. Nidhi Rastogi
Date: 15 December 2023

1. Data Science Problem

Introduction

In the world of machine learning, classification reigns supreme. It's the ability to categorize data points into predefined classes, a cornerstone of tasks like image recognition, spam filtering, and medical diagnosis. In some case studies, doing simple label classification is not enough; in such cases, multi-label classification is used. With the help of multi-label classification, we can classify news articles not just as 'sports' but also as 'breaking news' and 'international relations,' or a product description tagged as both 'eco-friendly' and 'tech-savvy.' This is the essence of multi-label classification: assigning multiple, non-mutually exclusive labels to a single data point.

Hence, the problem statement for this project is to "*Develop a multi-label classification model using the RCV1 dataset for accurate news article classification.*" The dataset is an archive of more than 800,000 manually categorized newswire stories made available by Reuters, Ltd

2. Data and Model Description

2.1 Dataset Description:

The Reuters Corpus Volume I (RCV1) is an archive containing over 800,000 manually categorized newswire stories made available by Reuters, Ltd. This dataset was fetched from Scikit Learn
Data can be described as following:

- 804,414 samples.
- 47,236 features (TF-IDF vectors)
- 103 labels
- Labels are stored in a CSR sparse matrix.
- Each sample has a value of 1 for the categories it belongs to, and 0 for others.

```
Shape of rcv1.data: (804414, 47236)
Shape of rcv1.target: (804414, 103)
rcv1.target_names: ['C11' 'C12' 'C13' 'C14' 'C15' 'C151' 'C1511' 'C152' 'C16' 'C17' 'C171'
 'C172' 'C173' 'C174' 'C18' 'C181' 'C182' 'C183' 'C21' 'C22' 'C23' 'C24'
 'C31' 'C311' 'C312' 'C313' 'C32' 'C33' 'C331' 'C34' 'C41' 'C411' 'C42'
 'CCAT' 'E11' 'E12' 'E121' 'E13' 'E131' 'E132' 'E14' 'E141' 'E142' 'E143'
 'E21' 'E211' 'E212' 'E31' 'E311' 'E312' 'E313' 'E41' 'E411' 'E51' 'E511'
 'E512' 'E513' 'E61' 'E71' 'ECAT' 'G15' 'G151' 'G152' 'G153' 'G154' 'G155'
 'G156' 'G157' 'G158' 'G159' 'GCAT' 'GCRIM' 'GDEF' 'GDIP' 'GDIS' 'GENT'
 'GENV' 'GFAS' 'GHEA' 'GJOB' 'GMIL' 'GOBIT' 'GODD' 'GPOL' 'GPRO' 'GREL'
 'GSCI' 'GSPO' 'GTOUR' 'GVIO' 'GVOTE' 'GWEA' 'GWELF' 'M11' 'M12' 'M13'
 'M131' 'M132' 'M14' 'M141' 'M142' 'M143' 'MCAT']
```

Figure 1: Summary of dataset.

Sparse Matrix: compresses the matrix data by storing only the non-zero elements and their corresponding row and column indices.

sparse.csr_matrix ((data, (rows, cols)), shape)

```
<804414x47236 sparse matrix of type '<class 'numpy.float64'>'
        with 60915113 stored elements in Compressed Sparse Row format>
```

Figure 2: Sparse matrix for rcv1 dataset.

2.2 Model Selection and Rationale:

While doing exploratory analysis it was seen that each sample belongs to more than one label. Hence for the scope of this project decided to go ahead with Multilabel Classification. The various classification models were explored like Linear SVM, Decision Tree, Random Forest Classifier, Naïve Bayes, and Linear Regression. The performance is evaluated on basis of hamming loss.

Hamming loss is used when performing multilabel classification. It Measures the fraction of labels that are incorrectly predicted. It is the ratio of the incorrect predictions to the total number of labels.

$$\text{Hamming Loss} = \frac{\text{Number of Incorrect Predictions}}{\text{Total Number of Labels}}$$

As the dataset is very large and complex the models are trained on small subsets starting with 1000 samples sets and then on 10,000 samples to see the model performance and evaluation. As trying to train and testing of entire dataset was leading to crashing of kernel.

3. Analysis Strategy

3.1 Preprocessing and Feature Engineering:

- RCV1 is already tokenized, containing non-zero values, cosine-normalized, log TF-IDF vectors; hence, not much preprocessing was required.
- The data split requirement was predefined: the first 23,149 samples for the training set and the last 781,265 samples for the testing set. However, due to limitations in Google Collab's capacity, a default split of 70% for training and 30% for testing was used in this project on smaller dataset of 1000 samples and later for 10000 samples.
- Given that the dataset was in the form of a sparse matrix, the initial preprocessing step involved converting the sparse matrix into a dense array using '.toarray()'.

```
# Convert y_train and X_train to dense arrays
y_train = y_train.toarray()
X_train = X_train.toarray()
# Convert y_test and X_test to dense arrays
X_test = X_test.toarray()
y_test = y_test.toarray()
✓ 0.4s                                                    Python
```

Figure 3: Code snippet of converting Sparse matrix to dense array.

- All the columns in training feature DataFrame that contain only zero values for all rows were identified. Once identified, it removes these columns from both the training and testing feature DataFrames.
- Similarly, columns from the target DataFrames were identified and removed where all values are zero.
- Through class Distribution graph in figure 4 it was known about the imbalance data distribution and hence Min Max Scaling was performed as a form of feature engineering.

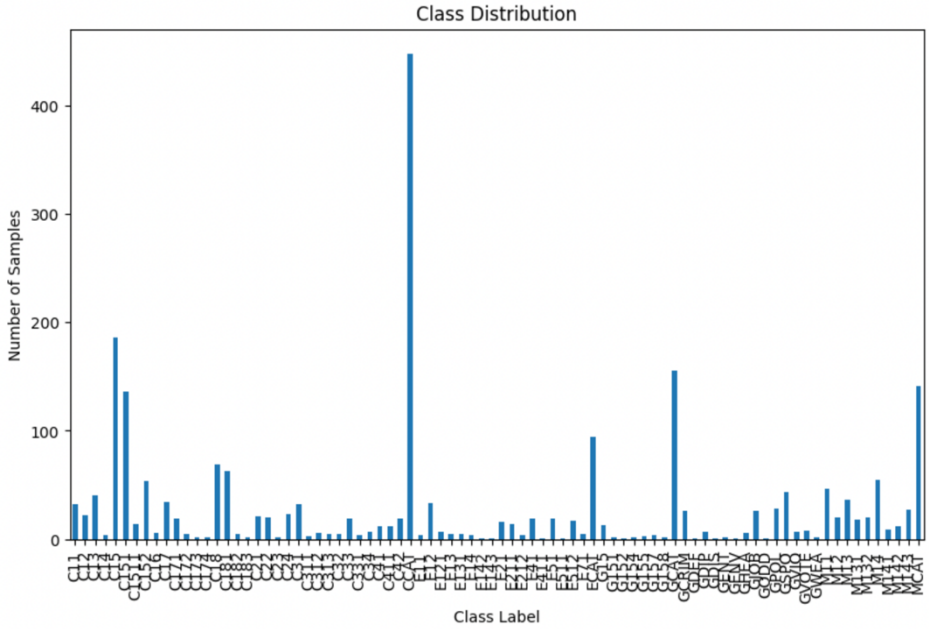

Figure 4: Class Distribution Diagram

3.2 Model Building and Regularization

- With the data being scaled, all models were trained, and their performance was evaluated.
- To ensure optimal model performance, Principal Component Analysis (PCA) was conducted, and models were retrained and tested.
- During model training, it was observed that when the sample set contained 1,000 records, the Decision Tree, although not delivering the best performance, ran smoothly. However, with a dataset of 10,000 samples, it took 90 minutes just to train the model. Consequently, due to its suboptimal nature, the model was dropped, and further analysis was conducted with the remaining four models.
- After performing PCA, gradient descent was employed during model training to ensure the models utilized the best parameters for optimum performance.
- Multinomial Naive Bayes is a probabilistic classification algorithm that typically doesn't use gradient descent for optimization. Instead, it estimates probabilities directly from the training data. The parameters in Multinomial Naive Bayes are probabilities associated with each term (feature) in each class.
- In the case of Random Forests, gradient descent isn't applicable as they have discrete hyperparameters (e.g., tree depth, number of trees, number of features). Additionally, the loss function may have multiple local minima, making any local search procedure susceptible to getting trapped. Therefore, only PCA was performed on the Random Forest Classifier.

3.3 Potential Challenges and Mitigation Strategies:

- Numerous challenges were encountered during the development of this project, with the primary issue arising from the size of the dataset. The Google Collab notebook consistently crashed even with just 10% of the dataset. Processing such a massive amount of data would take more than half an hour, only to crash midway due to exceeding RAM capacity.
- The conversion of a Sparse Matrix posed challenges. Several experiments were conducted to directly use the sparse matrix to train the models, but it resulted in errors.
- Assessing model compatibility was difficult, especially when working with a small sample of the entire dataset.

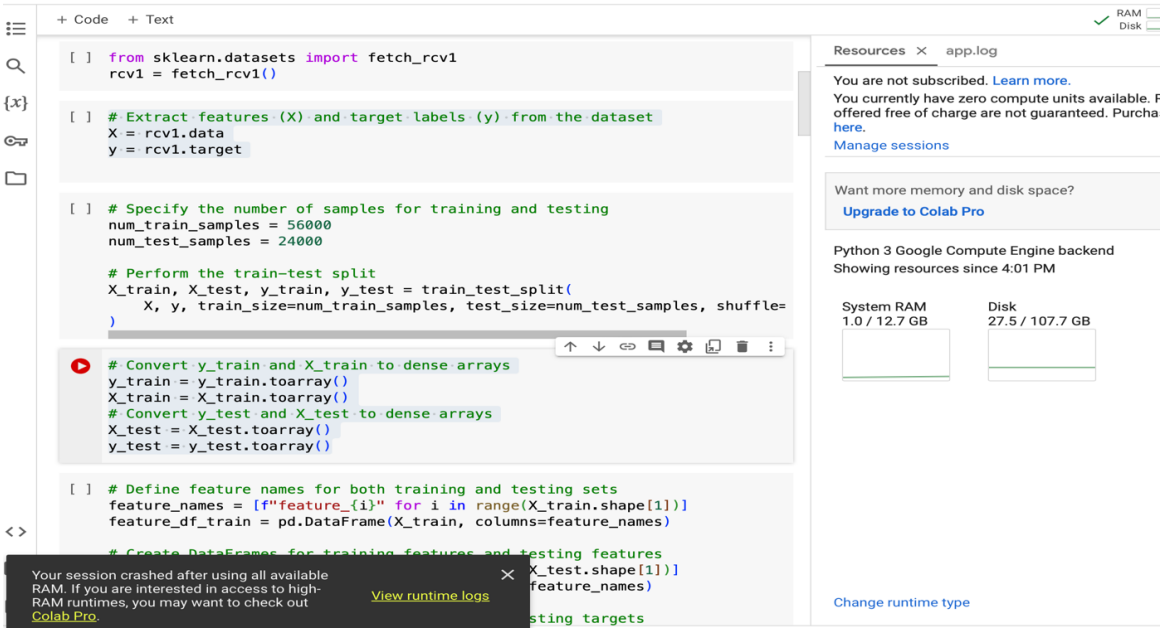

Figure 5: Screenshot of RAM crash While converting sparse to sense array for 80,000 samples

4. Analysis Code

4.1 Performance evaluation:

The model performance and observations are divided in 2 parts based on number of samples tested.

Observations on 1000 sample set:

1.Linear SVM: The performance of Linear SVM improved after PCA and gradient descent, achieving an accuracy of 12.67% and a hamming loss of 2.75%.

```
Metrics Performance for Linear SVM:
Accuracy: 12.00%
Hamming Loss: 2.83%
Weighted Precision: 77.46%
Weighted Recall: 34.33%
```

```
Metrics Performance for SVM with PCA and gradient descent  :
Accuracy: 12.67%
Hamming Loss: 2.75%
Weighted Precision: 81.42%
Weighted Recall: 34.01%
```

Code snippet 1: Performance of SVM.

2.Random Forest: In Random Forest, accuracy decreased, and hamming loss increased; PCA resulted in reduced model performance, with an accuracy of 1% and a hamming loss of 3.55%.

```
Metrics Performance for Random Forest:
Accuracy: 2.33%
Hamming Loss: 3.34%
Weighted Precision: 68.52%
Weighted Recall: 19.66%
```

```
Metrics Performance for Random forest with PCA :
Accuracy: 1.00%
Hamming Loss: 3.55%
Weighted Precision: 59.20%
Weighted Recall: 15.73%
```

Code snippet 2: Performance of Random Forest.

3.Logistic Regression: Logistic Regression showed improved performance after PCA and gradient descent, with an accuracy of 3.33% and a hamming loss of 3.14%.

```
Metrics Performance for Logistic Regression:
Accuracy: 0.00%
Hamming Loss: 3.89%
Weighted Precision: 41.84%
Weighted Recall: 10.63%
```

```
Metrics Performance for Logistic Regression with PCA and gradient descent :
Accuracy: 3.33%
Hamming Loss: 3.14%
Weighted Precision: 75.96%
Weighted Recall: 23.17%
```

Code snippet 3: Performance of Logistic Regression.

4.Naive Bayes: Naive Bayes demonstrated improved performance, achieving an accuracy of 7% and a hamming loss of 3.01%.

```
Metrics Performance for Naive Bayes :
Accuracy: 0.00%
Hamming Loss: 3.27%
Weighted Precision: 60.50%
Weighted Recall: 35.81%


Best Hyperparameters: {'alpha': 2.0}
Metrics Performance for Naive Bayse with hyper parameter tunning :
Accuracy: 7.00%
Hamming Loss: 3.01%
Weighted Precision: 75.71%
Weighted Recall: 28.48%
```

Code snippet 4: Performance of Naïve Bayes.

In comparison to all the above models, Linear SVM after PCA and gradient descent delivered the best performance with an accuracy of 12.67% and a minimum loss of 2.75% for 1000 sample set.

Observations on 10,000 sample set:

The observations were only performed on models after PCA and gradient descent to check their performance on higher subset of rcv1. As running all models was taking lot of time more that 90 minutes for each model.

1.Linear SVM: The performance of Linear SVM after PCA and gradient descent, achieving an accuracy of 51.93% and a hamming loss of 1.13%.

```
Metrics Performance for SVM with PCA and gradient descent  :
Accuracy: 51.93%
Hamming Loss: 1.13%
Weighted Precision: 89.26%
Weighted Recall: 74.67%
```

Code snippet 5: Performance of SVM.

2.Random Forest: In Random Forest with PCA resulted performance, with an accuracy of 11.03% and a hamming loss of 2.57%.

```
Metrics Performance for Random forest with PCA :
Accuracy: 11.03%
Hamming Loss: 2.57%
Weighted Precision: 89.29%
Weighted Recall: 25.38%
```

Code snippet 6: Performance of Random Forest.

3.Logistic Regression: Logistic Regression after PCA and gradient descent, with an accuracy of 5.63% and a hamming loss of 2.35%.

```
Metrics Performance for Logistic Regression with PCA and gradient descent :
Accuracy: 5.63%
Hamming Loss: 2.35%
Weighted Precision: 95.42%
Weighted Recall: 30.26%
```

Code snippet 7: Performance of Logistic Regression.

4.Naive Bayes: Naive Bayes demonstrated improved performance, achieving an accuracy of 25.27% and a hamming loss of 2.53%.

```
Best Hyperparameters: {'alpha': 2.0}
Metrics Performance for Naive Bayse with hyper parameter tunning :
Accuracy: 25.27%
Hamming Loss: 2.53%
Weighted Precision: 64.22%
Weighted Recall: 52.91%
```

Code snippet 8: Performance of Naïve Bayes.

In comparison to all the above models, Linear SVM after PCA and gradient descent delivered the best performance with an accuracy of 51.93% and a minimum loss of 1.13% for 10,000 samples.

4.2 Conclusion

- After performing model analysis in increasing fashion of sample subsets sets it was observed that Linear SVM with PCA and gradient descent performed best as compared with other models.
- Accuracy of each model increases with increasing in data subset size which indicate as model gets exposed to more possible combination of features resulting into target, the performance of model becomes better.
- PCA is helpful to increase performance.
- Gradient descent is not compatible with all models Eg. For naïve bayse is a probabilistic classification algorithm, and it typically doesn't use gradient descent for optimization. Instead, it works better with hyperparameter tunning.

4.3 Application

News aggregator app that allows users to filter articles based on their interests. By using multi-label classification, it can automatically categorize each article into multiple topics. This would enable users to quickly find articles that align with their specific preferences and interests.