Title: Model Implementation & Hyperparameter optimization using One Versus Rest Classifier with the estimator of Logistic Regression on Multi Label Dataset Problems, such as GoEmotion and DairAI

Author & The Course: Emre Erdoğan - COMP4602 Natural Language Processing

Many of the datasets that we come across in the Data Science field, such as the Go Emotion* (Reddit) and DairAI** (Twitter) datasets suffer from class imbalance, where some labels are heavily overrepresented while others are severely underrepresented. This imbalance can cause bias within the training procedure of Machine Learning (ML) / Deep Learning (DL) models. Especially in Multi Label Classification problems, a way to prioritise a more unbiased approach becomes highly important. Finding out whether or not balancing out the model fitting procedure or not was an academic relevance for the optimization of the model.

As mentioned before, both the Go Emotion* and the DairAI** Datasets have been used for this project. Go Emotion consists of 58.009 text examples each labeled with 27 + Neutral (28 in total) individual emotional labels, some of which may be labeled with more than one label. On the other hand, DairAI consists of 20.000 (16.000 Train, 2.000 Test, 2.000 Validation) text examples with simply 6 individual labeling options, which has similar label tags to Go Emotion dataset's labels. The similarity of the label names between two of these datasets makes aligning of the two datasets easier for future purposes. However, the labeling methods of both datasets greatly differ; While Go Emotion Dataset's labeling method rely on the tagging of the individual array indexes between 1 and 0, DairAI's dataset relies heavily on numerical tagging between the values of 0 to 5, which represents different emotional tagging. DairAI's labeling method works since it's a dataset that relies on multi class classification instead of multi label. However, in order to build an environment which supports both, preprocessing was required. Also, as mentioned in the research paper*, the fact that %83 of Go Emotion's dataset examples simply contained one emotional tagging, which means once again, when these two datasets were preprocessed and concatenated, were quite similar to each other.

Go Emotion Dataset Tables & Visuals:

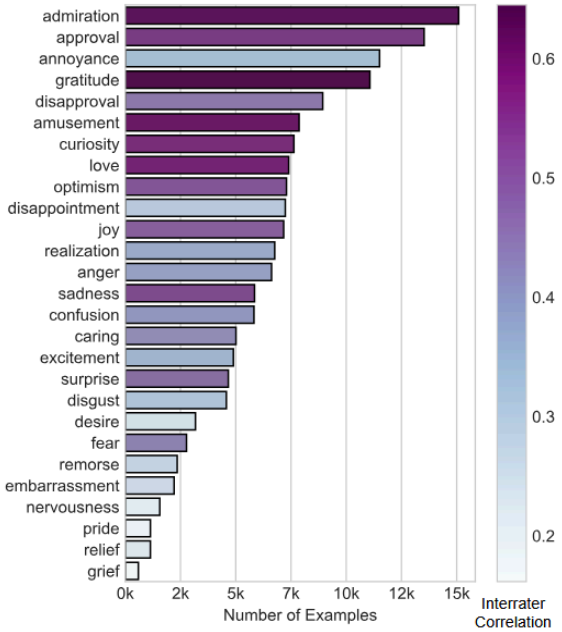


Figure 1: Our emotion categories, ordered by the number of examples where at least one rater uses a particu-

Number of examples	58,009
Number of emotions	27 + neutral
Number of unique raters	82
Number of raters / example	3 or 5
Marked unclear or difficult to label	1.6%
Number of labels per example	1: 83% 2: 15% 3: 2% 4+: .2%
Number of examples w/ 2+ raters agreeing on at least 1 label	54,263 (94%)
Number of examples w/ 3+ raters agreeing on at least 1 label	17,763 (31%)

Table 2: Summary statistics of our labeled data.

DairAI Dataset Tables & Visuals:

Data Fields

The data fields are:

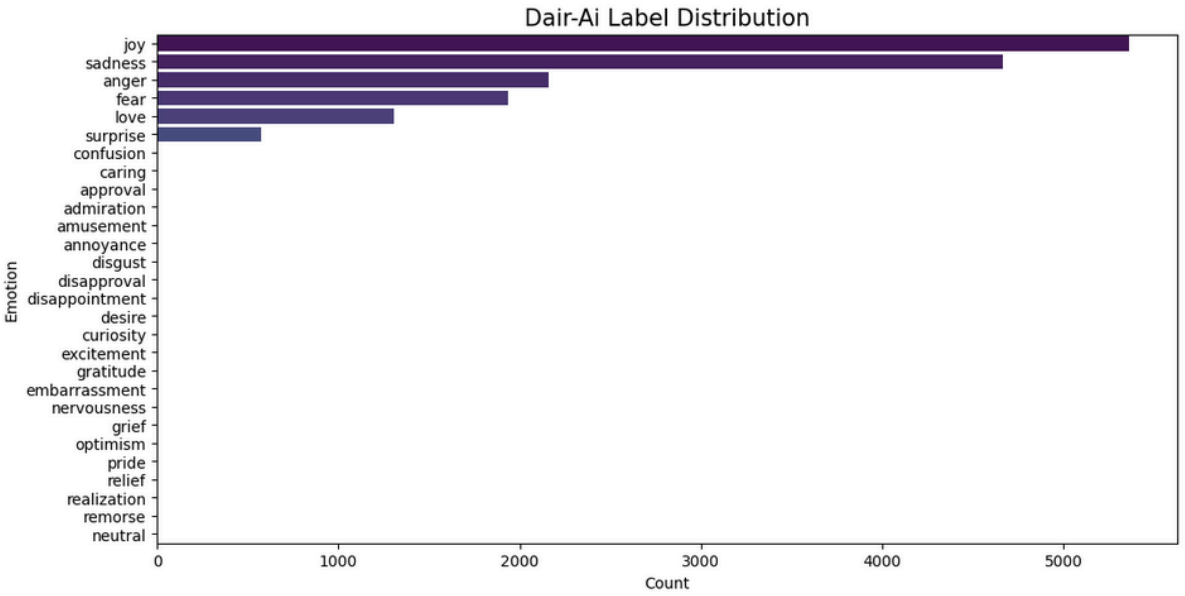
- `text`: a `string` feature.
- `label`: a classification label, with possible values including `sadness` (0), `joy` (1), `love` (2), `anger` (3), `fear` (4), `surprise` (5).

Data Splits

The dataset has 2 configurations:

- split: with a total of 20_000 examples split into train, validation and split
- unsplit: with a total of 416_809 examples in a single train split

name	train	validation	test
split	16000	2000	2000
unsplit	416809	n/a	n/a

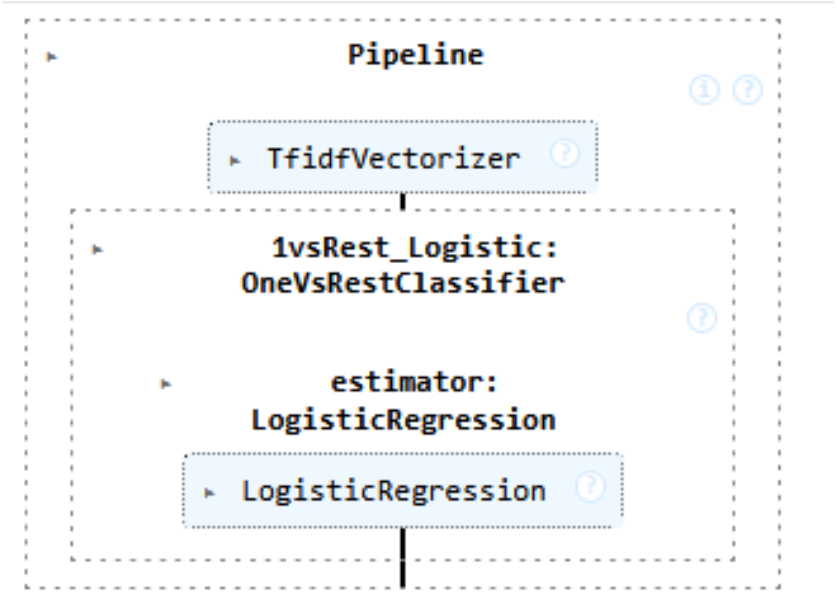


I first began my project by importing both the appropriate python libraries and the Go Emotion + DairAI Datasets. Since the labeling methodologies for both of these datasets were different, I created an algorithm that aligned the DairAI dataset's format into Go Emotion dataset's format, where each array index that represented an emotional label was tagged with the numerical values of either 1 or 0.

After which, I separated both the x (input text) and y (output emotional labels) for both datasets. I did this in order to define both the input and the output data which I planned on training my model with. Later, I moved over to splitting 80% of my x and y values of Go Emotion dataset's unsplit raw data as training data (x_train_goEmotion, y_train_goEmotion) and the remaining of the 20% as test data (x_test_goEmotion, y_test_goEmotion). Then, I concatenated both DairAI (x_train_dairAi, y_train_dairAi) and GoEmotion training data into a singular universal x_train and y_train.

After I was done with the preprocessing of these datasets, I moved over to the implementation of the pipeline for my NLP model. I first began by setting the pipeline for the baseline model, which I used TfidfVectorizer in order to vectorize each text example into a numerical vector. Then I added OneVsRestClassifier with the estimator of Logistic Regression for predicting each of the emotional labels one by one.

Following a similar pipeline for my hyperparameter optimized model, I used a RandomizedSearchCV method in order to find the best parameters for this pipeline. After going through different ngram ranges, min_df, max_df values for the TfidfVectorizer and different loguniform and class weight parameters for the OneVsRestClassifier that uses Logistic Regression as its estimator, I found the best parameters for obtaining the highest accuracy score for this model within my parameters range.



As a result of this project, I was able to increase the baseline accuracy score of 0.1335 into 0.1934. The reason behind such a low score was the foundational working methodology of the accuracy_score method on multi label datasets.

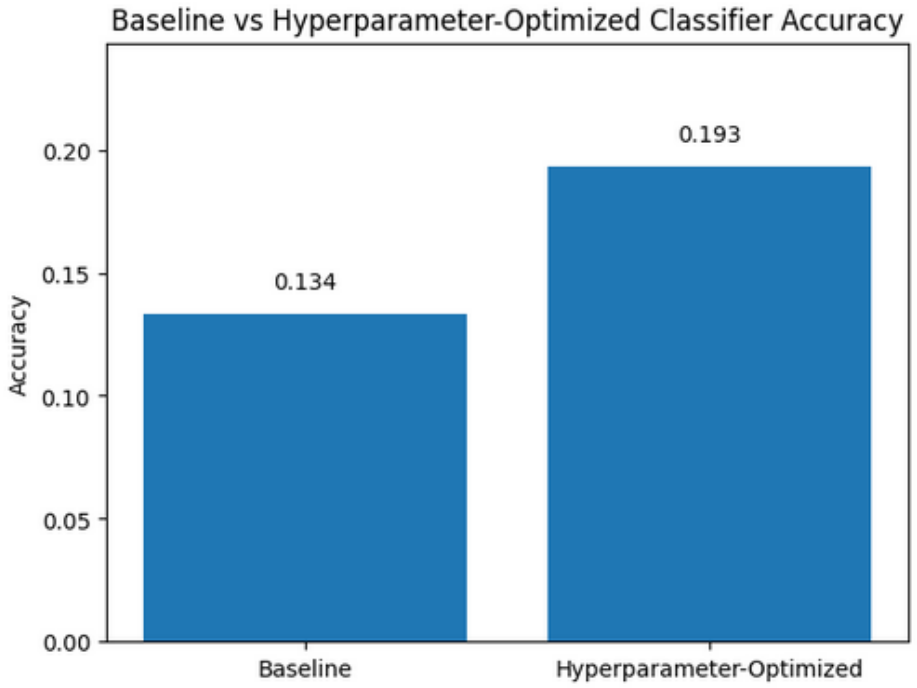
It was surprising for me to learn that the accuracy_score method expects the entire labels the same as the true values in order to count as correct. For example, if the text example's original labels are {Joy, Excitement} and if the model simply predicted {Joy}, the accuracy_score method would simply count the entire example as wrong. Ergo, showing the overall accuracy score of the entire model as low on multi label models even if it got most of the labels as correct. After research, I realized that accuracy_score is some of the harshest metrics methods for when working with multi label datasets.

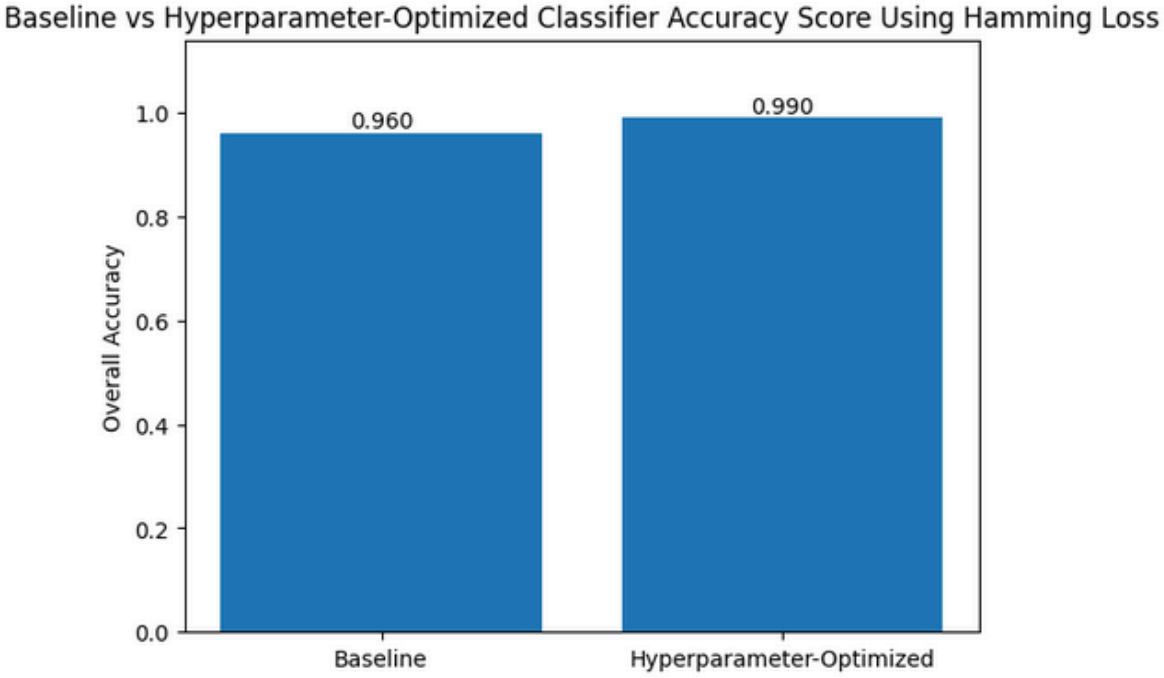
Therefore, I resorted to finding the overall accuracy score of my models by subtracting the hamming loss of my models with 1. An alternative to finding the overall accuracy instead of using the hamming loss would be using (y_true == y_pred).mean() function.

Using Hamming Loss calculations to measure the overall accuracy score showed an increase from 0.959851 to 0.989506 within the overall accuracy of the classification model.

Another surprising part was the fact that the best model scenario used mostly "None" as its class weight parameter, which means not resorting to balancing out the class imbalance during the training showed a greater increase within the dataset, which seemed odd to me.

	model	Accuracy	Overall Accuracy using Hamming Loss
0	Baseline	0.133508	0.959851
1	Hyperparameter-Optimized	0.193489	0.989506


Baseline vs Hyperparameter-Optimized Classifier Accuracy
Accuracy
0.00
0.05
0.10
0.15
0.20
0.134
0.193
Baseline
Hyperparameter-Optimized


Baseline vs Hyperparameter-Optimized Classifier Accuracy Score Using Hamming Loss
Overall Accuracy
0.0
0.2
0.4
0.6
0.8
1.0
0.960
0.990
Baseline
Hyperparameter-Optimized

Baseline accuracy: 0.1335

Baseline classification report:

	precision	recall	f1-score	support
admiration	0.68	0.26	0.37	3443
amusement	0.63	0.29	0.39	1921
anger	0.54	0.08	0.14	1619
annoyance	0.38	0.02	0.04	2701
approval	0.53	0.03	0.05	3465
caring	0.55	0.05	0.08	1195
confusion	0.55	0.04	0.07	1522
curiosity	0.52	0.05	0.10	1995
desire	0.45	0.05	0.10	771
disappointment	0.58	0.02	0.04	1674
disapproval	0.34	0.02	0.03	2338
disgust	0.63	0.07	0.13	1055
embarrassment	0.50	0.03	0.05	481
excitement	0.48	0.05	0.08	1122
fear	0.67	0.16	0.26	620
gratitude	0.92	0.70	0.80	2303
grief	0.33	0.01	0.01	139
joy	0.49	0.11	0.18	1594
love	0.70	0.34	0.45	1652
nervousness	0.42	0.01	0.03	356
optimism	0.60	0.15	0.24	1719
pride	0.57	0.01	0.03	274
realization	0.55	0.01	0.03	1794
relief	0.50	0.01	0.02	252
remorse	0.48	0.13	0.21	499
sadness	0.58	0.12	0.19	1336
surprise	0.52	0.10	0.16	1154
neutral	0.57	0.19	0.29	10919
micro avg	0.64	0.15	0.24	49913
macro avg	0.54	0.11	0.16	49913
weighted avg	0.56	0.15	0.21	49913
samples avg	0.17	0.16	0.16	49913

Test accuracy (best model): 0.1935

Optimized model classification report:

	precision	recall	f1-score	support
admiration	0.62	0.35	0.45	3443
amusement	0.60	0.38	0.46	1921
anger	0.45	0.17	0.25	1619
annoyance	0.31	0.07	0.11	2701
approval	0.30	0.07	0.12	3465
caring	0.37	0.10	0.15	1195
confusion	0.46	0.11	0.18	1522
curiosity	0.45	0.12	0.19	1995
desire	0.41	0.11	0.17	771
disappointment	0.27	0.04	0.08	1674
disapproval	0.31	0.07	0.12	2338
disgust	0.46	0.10	0.16	1055
embarrassment	0.41	0.10	0.16	481
excitement	0.36	0.10	0.15	1122
fear	0.54	0.22	0.31	620
gratitude	0.89	0.72	0.79	2303
grief	0.15	0.03	0.05	139
joy	0.43	0.19	0.26	1594
love	0.64	0.41	0.50	1652
nervousness	0.39	0.03	0.06	356
optimism	0.53	0.19	0.28	1719
pride	0.53	0.07	0.12	274
realization	0.29	0.06	0.09	1794
relief	0.12	0.02	0.03	252
remorse	0.47	0.19	0.27	499
sadness	0.48	0.20	0.28	1336
surprise	0.44	0.18	0.25	1154
neutral	0.52	0.33	0.40	10919
micro avg	0.52	0.22	0.31	49913
macro avg	0.44	0.17	0.23	49913
weighted avg	0.47	0.22	0.29	49913
samples avg	0.25	0.24	0.24	49913

I could safely say that I was able to learn how the fundamentals of pipelines worked, how to implement pipelines for my projects, and how to hyperparameter optimize ML/DL models using various estimators and parameters. Overall, it was personally a great learning experience for my career.

I plan on experimenting with different kinds of machine learning models, and even perhaps deep learning models to better optimize my model for the future.

*GoEmotions: A Dataset of Fine-Grained Emotions - https://aclanthology.org/2020.acl-main.372.pdf
**DairAi - https://huggingface.co/datasets/dair-ai/emotion
GoEmotions: A Dataset for Fine-Grained Emotion Classification - https://research.google/blog/goemotions-a-dataset-for-fine-grained-emotion-classification/