Simple yet efficient approach for text classification

Feature Engineering

The following steps were applied to the dataset in the given order:

1. We concatenated all of the features together to get the most out of the information given to us.
2. We removed all of the duplicated rows to avoid overfitting.
3. Data points having class labels with counts less than 10 were removed because it shouldn't affect our score that much as we're monitoring accuracy.
4. Label normalization was done where labels were mapped to values from 0 to 9918 because we found that the BROWSE_NODE_ID were not continuous.

Approach

- Since we had a huge number of data points and the dimensionality was also a challenging issue, we thought of not diving into deep learning as its non-trivial.
- We tried a couple of combinations of linear classifiers that best fit the sparse matrix like SGD, Multinomial Naive Bayes, Passive-Aggressive classifiers etc and found that Passive-Aggressive classifiers work best in the case of given data.
- We used Hashingvectorizer for word embeddings and fixed the dimensionality as 2**18 as other combinations didn't work out well.
- We tried other vectorizers also like tf-idf with SVD for dimensionality reduction but couldn't do well.
- We feed the embeddings vector in a batch size of 40000 and ensure that at least all the training points pass through the model once.
- We split the training data in the ratio of 99.05 and 0.5. And the accuracy was around 75% and that is a well balanced score for a linear classifier.

Tools

- Pandas, Numpy, Scipy, Scikit-learn, NLTK,