AI511 ML Project Report

Project Name: Help Boost Our Online Reach!

Team Name: GI Joe

Team Members:

Ishan Shanware - IMT2019037
Ghazi Shazan Ahmad - IMT2019033

Contents

Project Definition

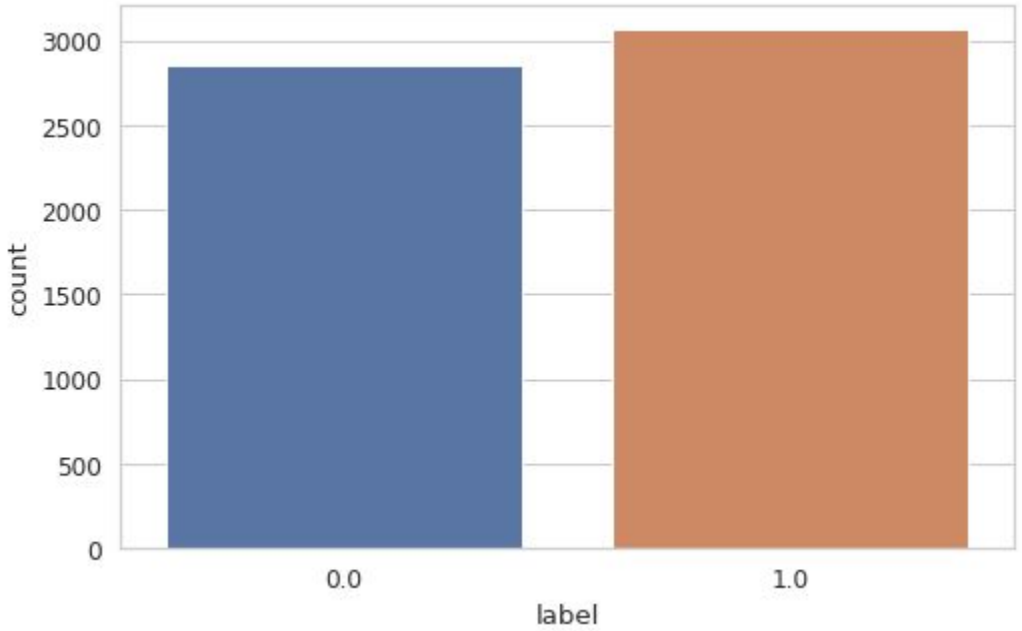
- The aim of this task is to identify the relevant, high-quality web pages from a pool of user-curated web pages, for the identification of “ad-worthy” web pages.

- The challenge requires us to build large-scale, end-to-end machine learning models that can classify a website as either “relevant” or “irrelevant”, based on attributes such as alchemy category and its score, meta-information of the web pages and a one-line description of the content of each webpage

Exploratory Data Analysis

Basic Introduction to Data For Non-Text Features

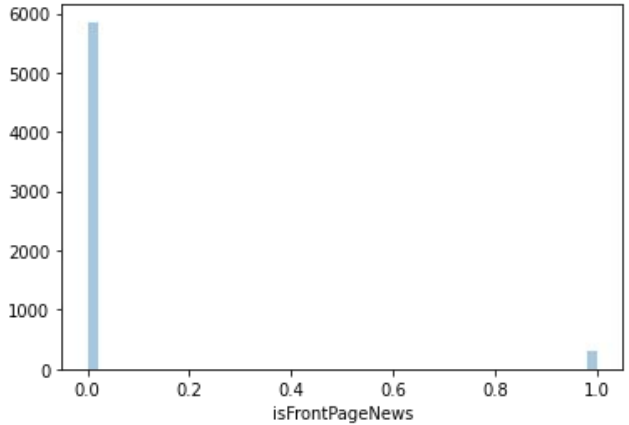
- There are both categorical and non-categorical columns present.
- Categorical Data was divided into binary class data (like isNews, isFrontPageNews etc.) and data with multiple categories like alchemy_category.
- We worked on a Dataframe by concatenating the train data and test data. So that we can perform preprocessing on both test and train data together. Total rows of Train Data - 5916 and Total rows of Test Data - 1479. Total rows of combined data frame - 7395
- There were 4 fields with missing values (i.e, '?'):
    - Alchemy_category_score - Numerical Class - 2342 Missing values
    - Alchemy_category - (Categorical class (multiple classes)) - 2342 Missing values
    - isFrontPageNews - Binary Class - 1248 Missing Values
    - isNews - Binary Class - 2843 Missing Values.
- We observed that rows with a missing values for alchemy_category also don't have a alchemy_category_score.

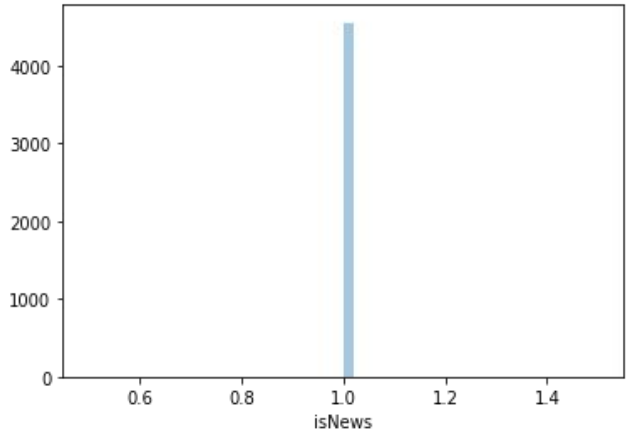


Labels are pretty much balanced

Dealing with Missing Values - 1

- Missing values in isFrontPageNews: We observed that the mode is 0, i.e, most rows had value 0 for isFrontPageNews. So we replaced the missing values for that column with 0. Below is the histplot of the values of the column.

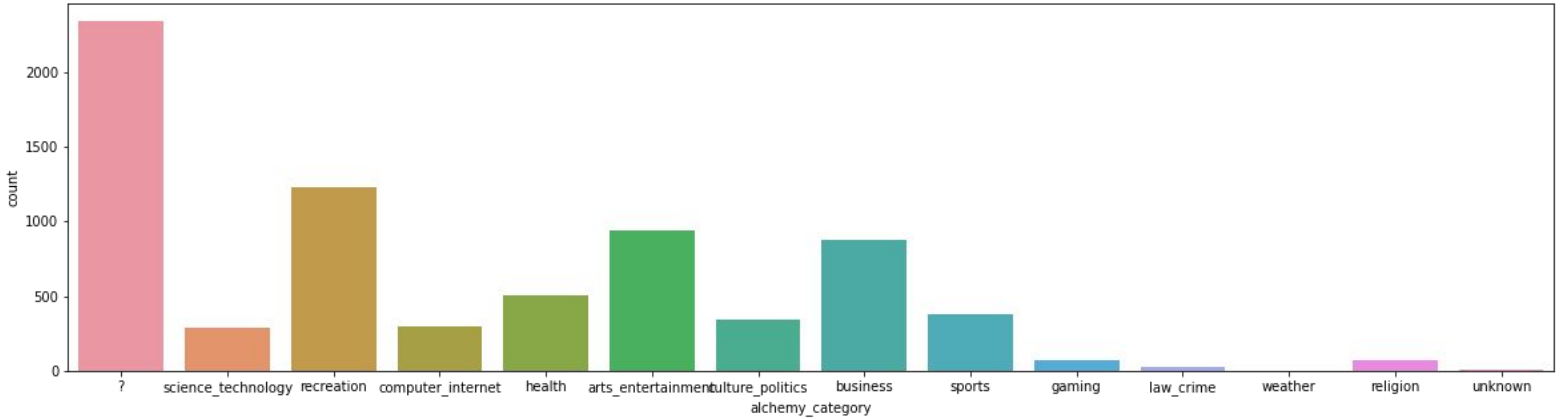




- Missing values in isNews: We observed that the mode for is 1, i.e, most rows had value 1 for the isNews column. The rows where isFrontPageNews was 1 and the isNews was missing, we replaced the isNews with 1 as obviously if a website contains front page news then it is a news website. For other missing values in isNews with 0.

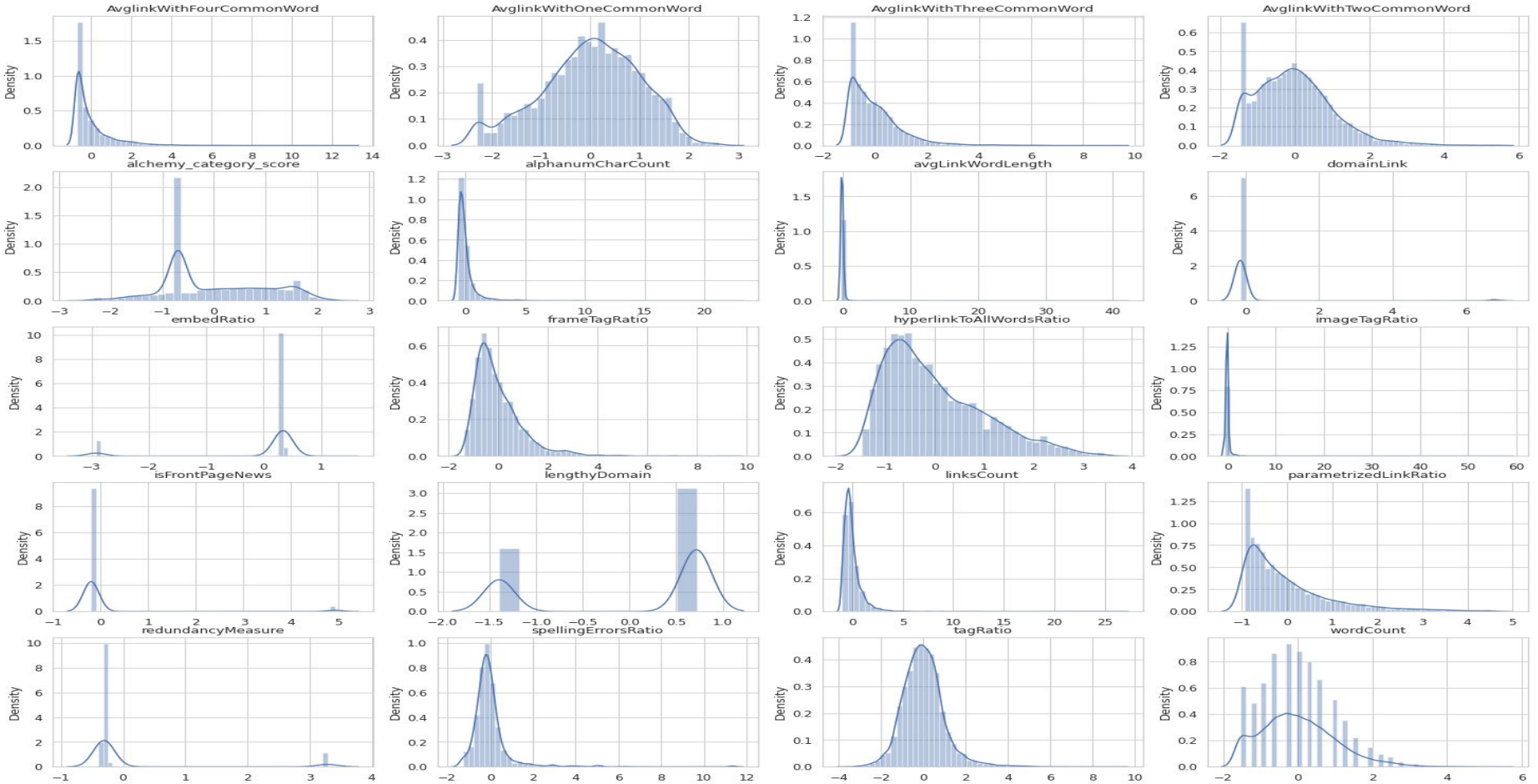
Dealing with Missing Values - 2

- Missing values in alchemy_category: We observed that maximum rows have '?' as compared to any category, for the alchemy_category feature. We decided to replace, '?' with 'unknown'. Below is the histplot for the feature alchemy_category.



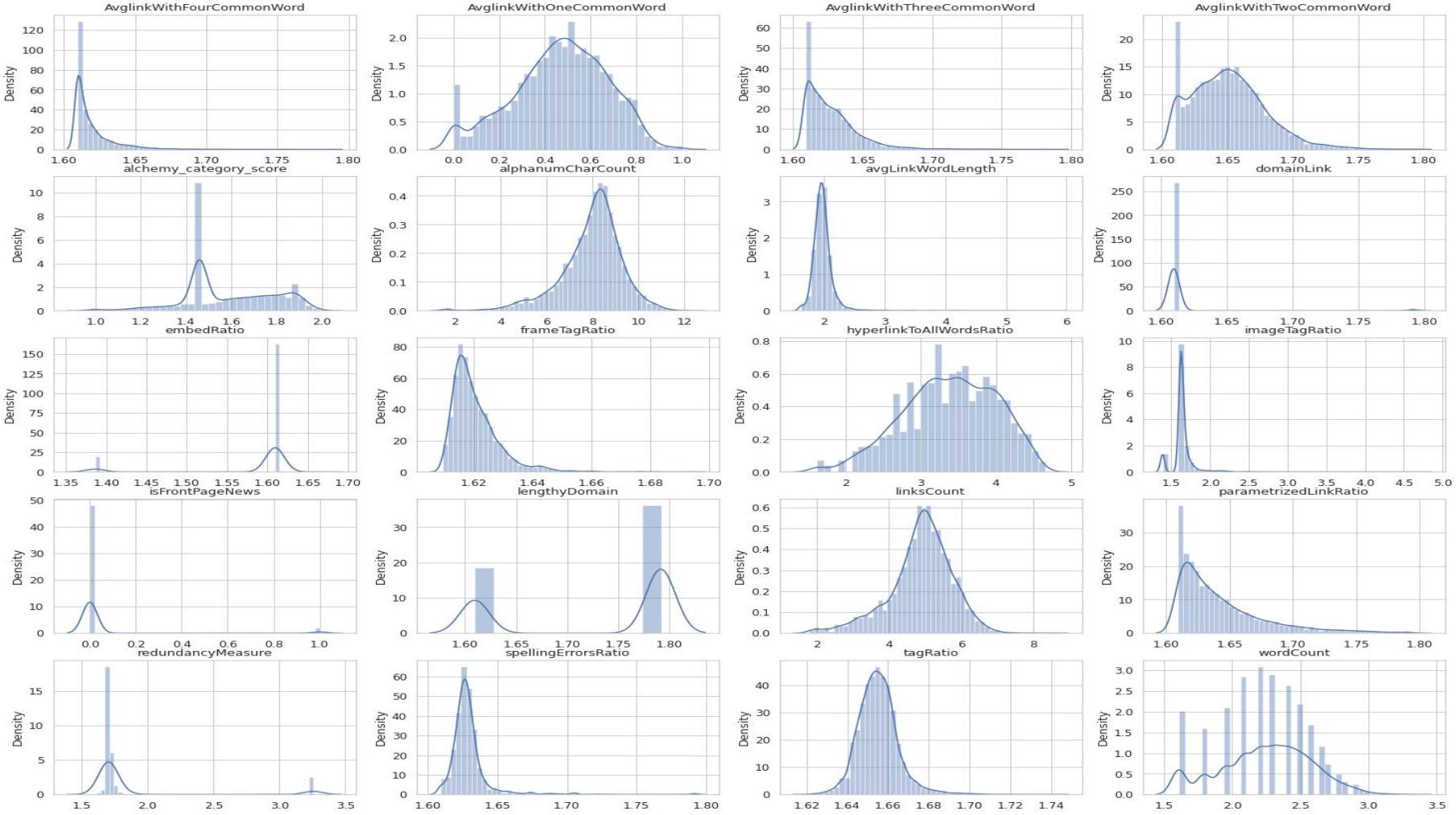
- Missing values in alchemy_category_score: We observed that all values for alchemy_category_score were 0.400001 for rows which had 'unknown' for alchemy_category. So we replaced all '?' in the alchemy_category_score feature with 0.400001, since they have 'unknown' for alchemy_category feature. We also converted the datatype of the alchemy_category_score to float64. This was a good idea because alchemy_category_score had no outliers after this replacement.

Dealing with skewed Data (below are the distplots with kde for the numerical data)

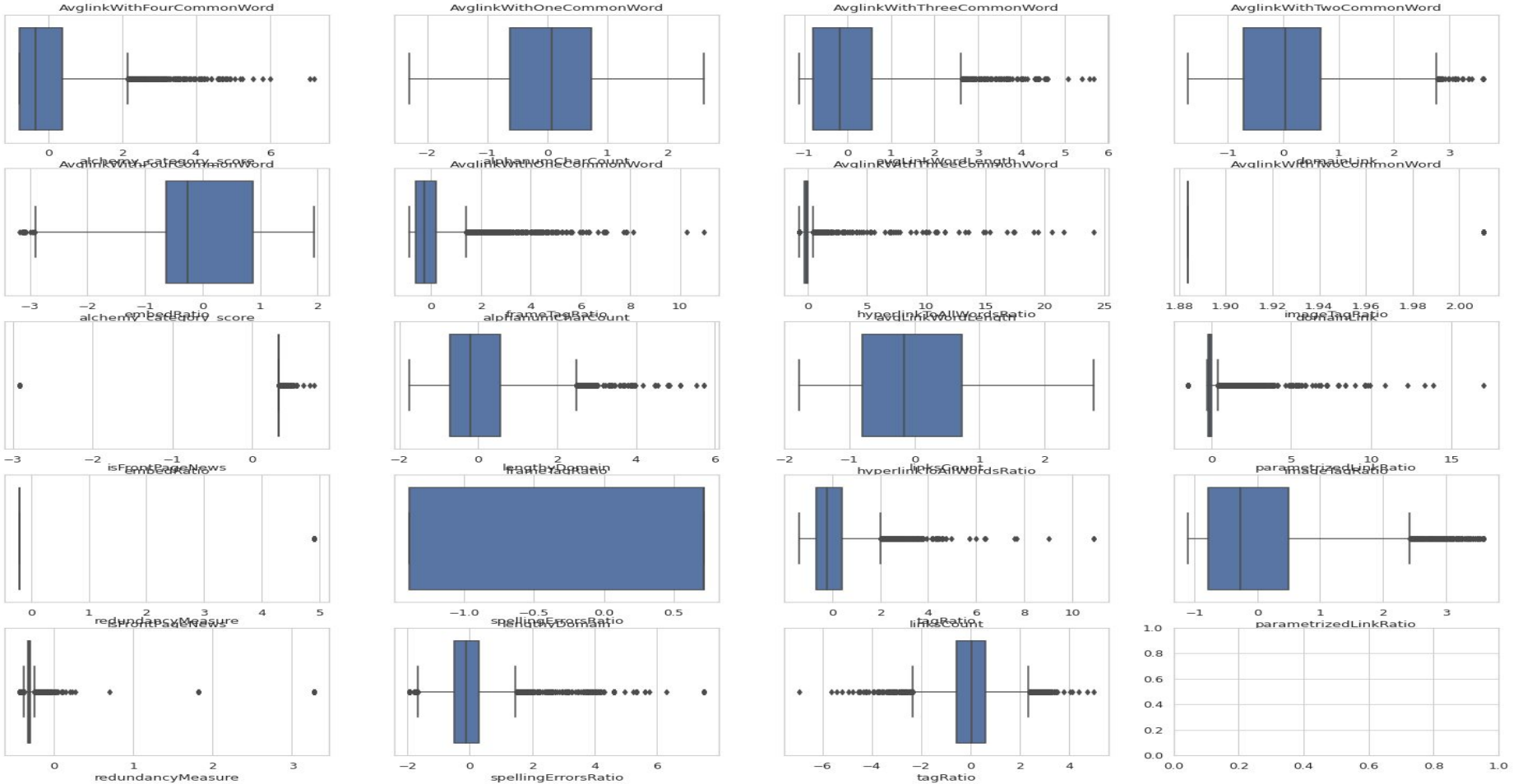


Dealing with skewed Data (continued....)

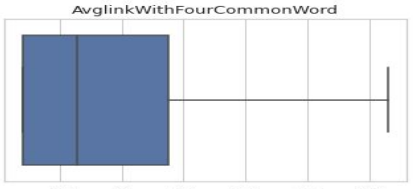
- The Previous slide contains the distplots for all numerical features. We can see that most data is skewed towards the right.
- However, most data which is skewed towards the right also contains large number of zeroes.
- Inorder , to deal with this we added a very small value of about 5 to the features with large number of zeroes skewed towards the right.
- Then we applied log transformation to fix the skew.
- Since, most data have different scales(for example some features are a type of count, others are ratios), we decided to normalize the data.
- Next slide contains plots after removing skews.

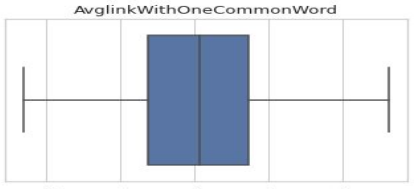

AvglinkWithFourCommonWord
AvglinkWithOneCommonWord
AvglinkWithThreeCommonWord
AvglinkWithTwoCommonWord
alchemy_category_score
alphanumCharCount
avgLinkWordLength
domainLink
embedRatio
frameTagRatio
hyperlinkToAllWordsRatio
imageTagRatio
isFrontPageNews
lengthyDomain
linksCount
parametrizedLinkRatio
redundancyMeasure
spellingErrorsRatio
tagRatio
wordCount
Density

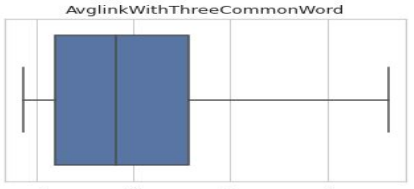
Removing outliers from train data (So most features in train data, contained huge outliers)

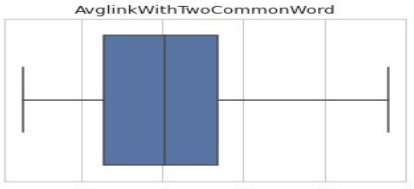


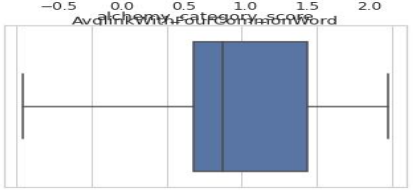
In order to remove these outliers, we simply substituted outliers with upper/lower whiskers of the boxplots.

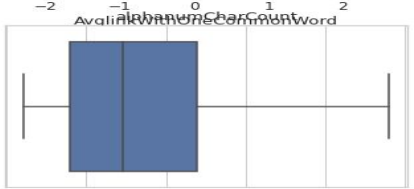


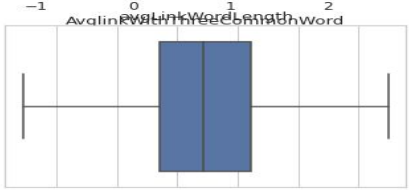


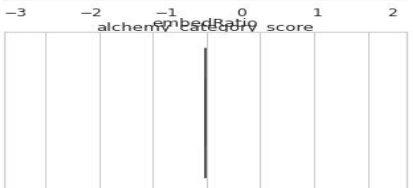


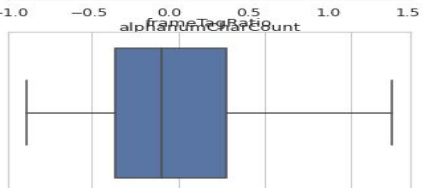


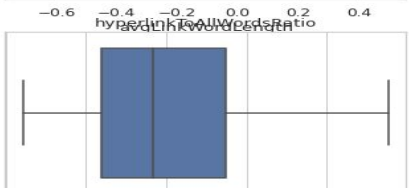


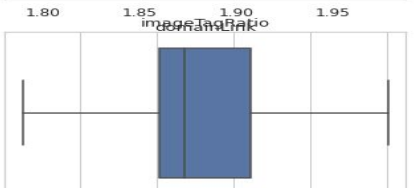


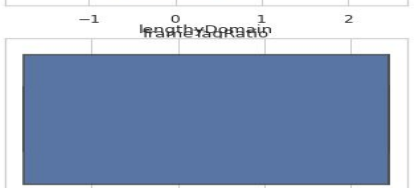


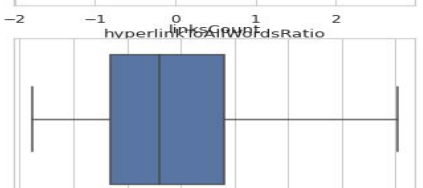


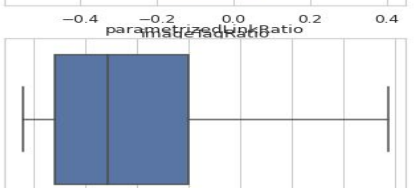


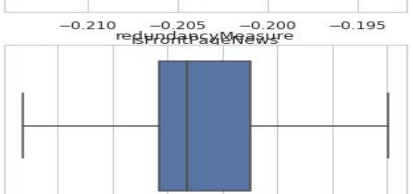


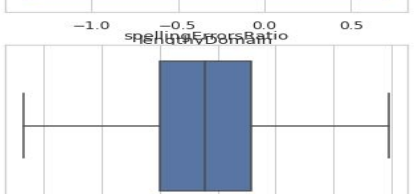


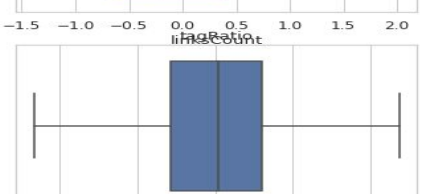


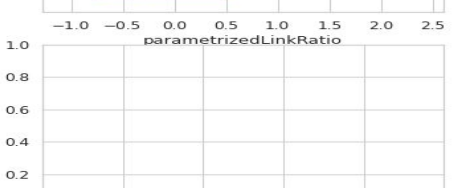












Text Preprocessing - 1

- There are only two columns with text values in train CSV file
  - Url - link to the website
  - WebpageDescription - Json String containing:
    - Body
    - Title
    - URL
    - Related
- The feature url isn't of much use as the text present in url is already present in the feature webpageDescription.
- In order to convert the Json String to text Features we used the json parser which comes with the json python library.
- We then converted the json to text and concatenated the Dataframe with the main dataframe.
- To deal with the missing values of the in body, title features obtained after json parsing, we applied the following 2 methods:
  - Replace the empty values, with empty strings in rows where body or title are missing.
  - Combine the features to body, title features. This method is effective than the previous one, as it also deals with the words which occur in both features, thus reducing the size of hte vocabulary.

Text Preprocessing - 1 (NLP)

For Evaluation 1

- There were some rows which had null body or title fields.
- We replaced those null values with empty strings denoting that there are no words associated of that field for the web page.
- Cleaned the text, by removing punctuations.
- Tokenized the text, and converted the text to a list of words.
- Removed stopwords from the list as they are unnecessary and don't provide useful information.
- Lemmatized the list of words which converts words to its root form(word).

For Evaluation 2

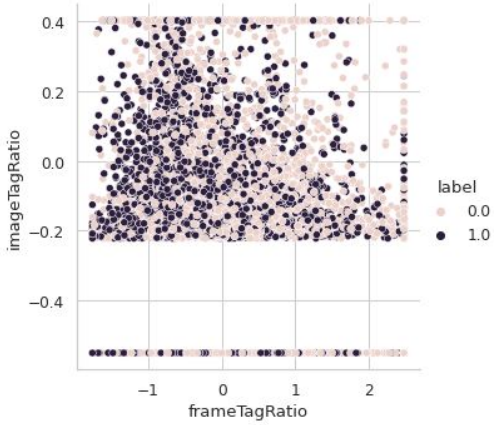
- For the webpageDescription, we tried combining title and body text. We thought this is a good idea, since there were many common words between the 2 types of string after stemming.
- Therefore, combining the 2 columns, eliminates some common words.
    - We made about 5 models, using the tfidf embedding implemented over the column,
    - However the results still didn't beat out best kaggle submission.

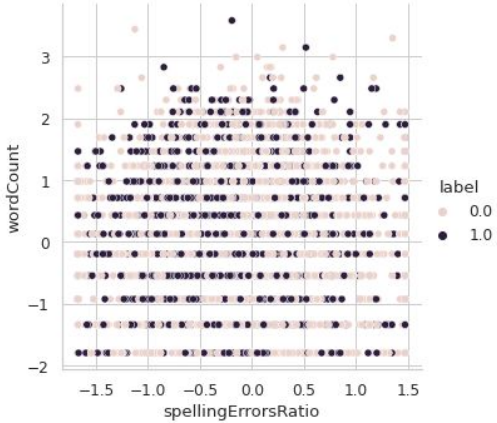
Text Preprocessing - 2 (HTML Content)

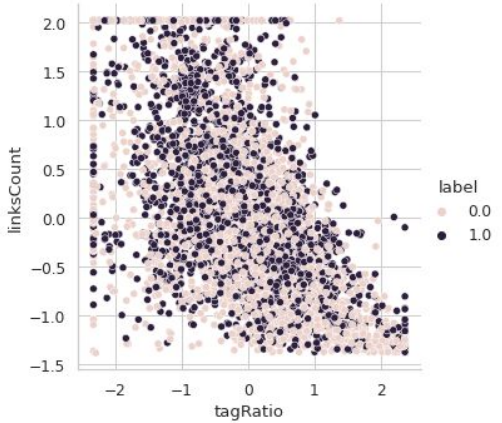
- We parsed HTML content files using beautiful soup html parser.
- The html parser removes all html tags from the data and return the text.
- We then break the text into lines, and remove leading and trailing spaces from each line.
- Nested lines are broken into lines again.
- After, obtaining lines we concat them with a space between each line, and obtain the text from it.
- After, this perform the usual text preprocessing.
    - Removing punctuation
    - Remove Stopwords
    - Remove spaces, converting to list
    - Stemming
- Using, tfidf we were getting a vocabulary size of about 1 lack words.
- However, by using the min_df parameter we were able to convert the vocab size to about 18-30k, depending on how big the min_df param is.

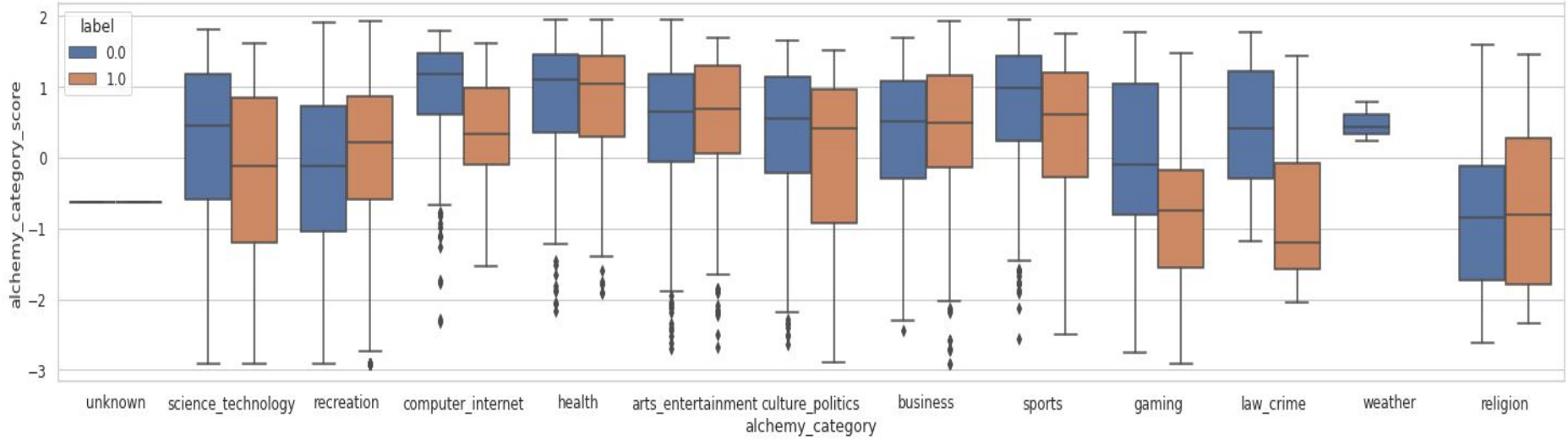
Exploratory Data Analysis on Numerical Features

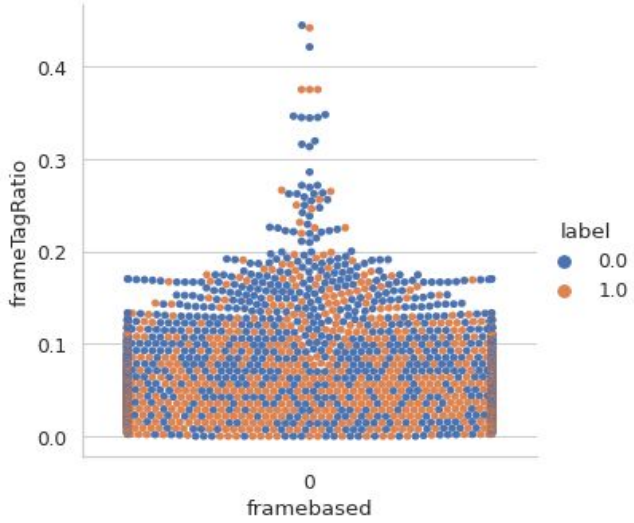
- In EDA, we made seaborn plots to draw relations between categorical features and numerical features.
- Some, scatterplots and relplots helped us see that all rows for features isNews and Framebased were the same. Therefore, there was not much unique data for the features, so we decided to drop the features.
- We were trying to see if the values alchemy_category_score is same for a type in alchemy_category. If this was true, we would be able to convert the feature alchemy_category_score to a categorical column by binning.
- However, the violinplots showed that the alchemy_category_scores for all other categories in alchemy_catgory varied a lot.
- Boxplots between alchemy_category and alchemy_category_score showed that most high traffic pages have low alchemy_category_scores. (Plots on next slide)
- Countplots from alchemy_category_score showed that ‘unknown’ category had the max frequency.
- Relplots between imagetagRatio and frameTagRation didn’t give any interesting observations.
- Correlation plot, showed high correlation for all avgLinkwithCommonWord type features.
- Correlation plot, showed that most non text features didn’t have really high correlations.
- tagRatio and linkCount also have good level of correlation, the same is seen in the relplot between them.

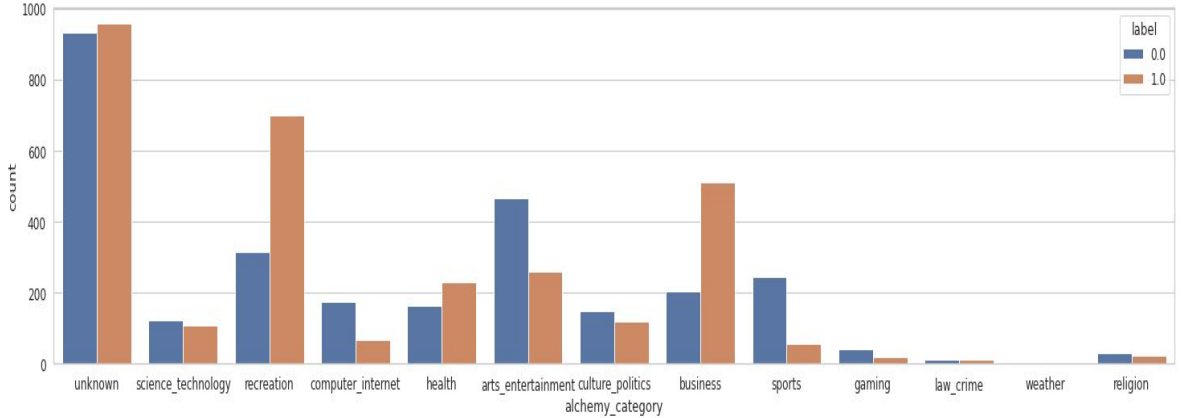

imageTagRatio
frameTagRatio
label
0.0
1.0

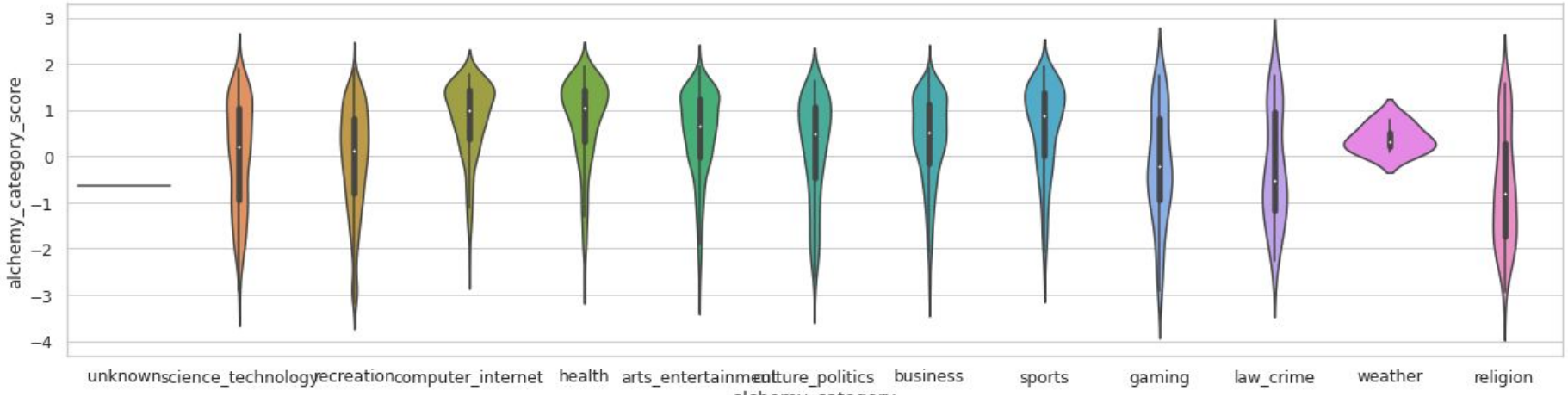

wordCount
spellingErrorsRatio
label
0.0
1.0

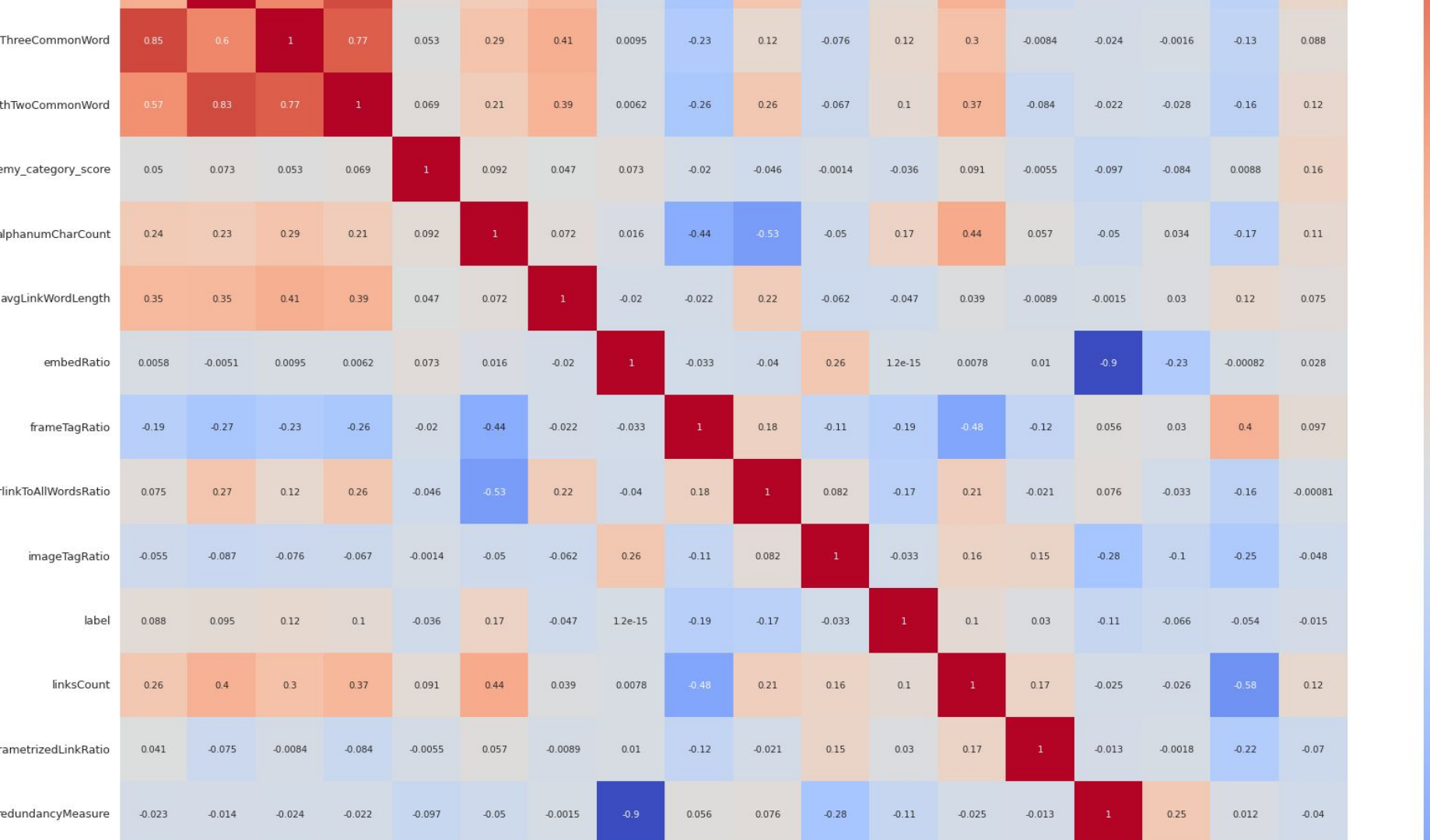

linksCount
tagRatio
label
0.0
1.0


label
0.0
1.0
alchemy_category_score
unknown
science_technology
recreation
computer_internet
health
arts_entertainment
culture_politics
business
sports
gaming
law_crime
weather
religion
alchemy_category


frameTagRatio
0.4
0.3
0.2
0.1
0.0
label
0.0
1.0
0
framebased


count
1000
800
600
400
200
0
label
0.0
1.0
unknown
science_technology
recreation
computer_internet
health
arts_entertainment
culture_politics
business
sports
gaming
law_crime
weather
religion
alchemy_category


alchemy_category_score
3
2
1
0
−1
−2
−3
−4
unknown
science_technology
recreation
computer_internet
health
arts_entertainment
culture_politics
business
sports
gaming
law_crime
weather
religion


ThreeCommonWord
thTwoCommonWord
emy_category_score
lphanumCharCount
avgLinkWordLength
embedRatio
frameTagRatio
rlinkToAllWordsRatio
imageTagRatio
label
linksCount
rametrizedLinkRatio
redundancyMeasure
0.85 0.6 1 0.77 0.053 0.29 0.41 0.0095 -0.23 0.12 -0.076 0.12 0.3 -0.0084 -0.024 -0.0016 -0.13 0.088
0.57 0.83 0.77 1 0.069 0.21 0.39 0.0062 -0.26 0.26 -0.067 0.1 0.37 -0.084 -0.022 -0.028 -0.16 0.12
0.05 0.073 0.053 0.069 1 0.092 0.047 0.073 -0.02 -0.046 -0.0014 -0.036 0.091 -0.0055 -0.097 -0.084 0.0088 0.16
0.24 0.23 0.29 0.21 0.092 1 0.072 0.016 -0.44 -0.53 -0.05 0.17 0.44 0.057 -0.05 0.034 -0.17 0.11
0.35 0.35 0.41 0.39 0.047 0.072 1 -0.02 -0.022 0.22 -0.062 -0.047 0.039 -0.0089 -0.0015 0.03 0.12 0.075
0.0058 -0.0051 0.0095 0.0062 0.073 0.016 -0.02 1 -0.033 -0.04 0.26 1.2e-15 0.0078 0.01 -0.9 -0.23 -0.00082 0.028
-0.19 -0.27 -0.23 -0.26 -0.02 -0.44 -0.022 -0.033 1 0.18 -0.11 -0.19 -0.48 -0.12 0.056 0.03 0.4 0.097
0.075 0.27 0.12 0.26 -0.046 -0.53 0.22 -0.04 0.18 1 0.082 -0.17 0.21 -0.021 0.076 -0.033 -0.16 -0.00081
-0.055 -0.087 -0.076 -0.067 -0.0014 -0.05 -0.062 0.26 -0.11 0.082 1 -0.033 0.16 0.15 -0.28 -0.1 -0.25 -0.048
0.088 0.095 0.12 0.1 -0.036 0.17 -0.047 1.2e-15 -0.19 -0.17 -0.033 1 0.1 0.03 -0.11 -0.066 -0.054 -0.015
0.26 0.4 0.3 0.37 0.091 0.44 0.039 0.0078 -0.48 0.21 0.16 0.1 1 0.17 -0.025 -0.026 -0.58 0.12
0.041 -0.075 -0.0084 -0.084 -0.0055 0.057 -0.0089 0.01 -0.12 -0.021 0.15 0.03 0.17 1 -0.013 -0.0018 -0.22 -0.07
-0.023 -0.014 -0.024 -0.022 -0.097 -0.05 -0.0015 -0.9 0.056 0.076 -0.28 -0.11 -0.025 -0.013 1 0.25 0.012 -0.04

Experiments conducted and challenges faced (Including Resource Management)

- We worked extensively on extracting text from html_content, we were using this text to train our model (though it was overfitting).
- Process extracting Text:
  - We parsed HTML content files using beautiful soup html parser.
  - The html parser removes all html tags from the data and return the text.
  - We then break the text into lines, and remove leading and trailing spaces from each line.
  - Nested lines are broken into lines again.
  - After, obtaining lines we concat them with a space between each line, and obtain the text from it.
  - After, this perform the usual text preprocessing.
    - Removing punctuation
    - Remove Stopwords
    - Remove spaces, converting to list
    - Stemming
    - Using, tfidf we were getting a vocabulary size of about 1 lack words.
    - However, by using the min_df parameter we were able to convert the vocab size to about 18-30k, depending on how big the min_df param is.

Experiments conducted and challenges faced (continued……..)

- Results of Experiments with Html Data: Most Models were strangely overfitting.
- Scores were dropping a lot when submitted on kaggle.
- We were getting scores of about 0.49, which was clearly indicating overfitting.
- Logistic Regression:
    - ○ Logistic Regression was producing overfitting models, no matter how many iterations we give.
    - SVM
        - ○ We decided to use PCA before, implementing any SVM kernel since the vocab is quite huge.
- In the process of extracting the text from the html_content (which takes roughly an hour), google colab was crashing a lot, so we switched to kaggle.
- Reasons being:
    - PCA was often crashing colab, we need to work out a way to avoid this.
    - Processing all HTML files is computationally heavy, colab sometimes crashes in between.
- AWS, RAM was insufficient we switched to kaggle which provided 16gb ram.
- Kaggle also turned to be extremely stable as compared to google colab, which crashed a lot while training RBF kernels.

Feature Selection and Engineering

- As we had around 70000 features, many models took too long to train.
- we used Principal Component Analysis to to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the data set.
- after doing PCA, the training time reduced considerably.
- Played around with different hyperparameters
- Got 1-3 k n_features as the most appropriate number of features to train.
- Numerical features didn’t give good results, therefore we resorted to textual features.

Different Learning algorithms

- Logistic Regression:
  - Logistic Regression was producing overfitting models, no matter how many iterations we give.
- SVM
  - We decided to use PCA before, implementing any SVM kernel, since the vocab is quite huge.
  - PCA was often crashing colab, we need to work out a way to avoid this.
  - Processing all HTML files is computationally heavy, colab sometimes crashes in between.
  - We were not able to try on kaggle due to time constraint
  - We used linear, rbf, polynomial kernels to train models.
- XGBoost
  - XGBoost also gives an overfitting model, we were setting gamma on auto.
    Will implement more models soon:)
- Adaboost
- RandomForestClassifiers
  - We use GridSearchCV to obtain the optimal hyperparameters.

Walkthrough of Best Kaggle Submission

- Text Preprocessing:
  - Remove stopwords
  - Remove punctuation
  - Remove spaces
  - Concat Strings
  - Remove trailing and Leading spaces
  - Stemming and not Lemmatization.
- After this, we converted stemmed words to a tfidf matrix, with min_df = 3.
- Since, features were a lot. We reduced features using PCA to about 3000.
- Then we used SVM, with ‘rbf’ kernel.

Training time:

Kaggle Score:
0.88815

Models Using Other Learning Algorithms and Experiment with different Hyperparameters

- Logistic Regression
- SVM with Linear Kernels (PCA before SVM)
- SVM with ‘rbf’ kernel (PCA before SVM)
- Random Forest Classifier
- XG Boost
- ADA Boost

We played around with different hyperparameters, to try and increase the scores.
This was fun, as we learnt Decision trees and Boosting recently. But a lot of work is still left.

Next Slides contain a details table of the different models implemented by us.

All scores are roc_auc_socres

Logistic Regression

- Training time: 2 minutes approx
- Score: 0.88515
- Hyperparameters: max_iter = 10000 and used predict_proba

SVM with ‘linear’ kernel

- Training time: without using PCA, it would take hours and hours; after reducing the dimensions to a few thousands, it took around 3-4 minutes

- Score: 0.88815(our best score till now)

- Hyperparameters: C = 0.5, enabled probability estimates

SVM with ‘rbf’ kernel

- Training time: without using PCA, it would take hours and hours; after reducing the dimensions to a few thousands, it took around 3-4 minutes

- Score: 0.88815(our best score till now)

- Hyperparameters: C = 0.5, enabled probability estimates

Random Forest Classifier

Training Time = Atleast an Hour

We played around with the n_components Hyperparameter

We obtained Good Results with n_components = 500, Training time was approximately 3hrs.

Test Data: auc_score = 0.8532 (Since, this was lesser than our best kaggle submission we did not submit on kaggle)

XGboost

- Training time: few minutes
- Score: 0.88621 on Kaggle
- No particular hyperparameters

Adaboost

- Training time: 2 minutes
- Score: 0.86001
- Hyperparameters: Not particular

Models built over webpage Description

Model Number	Training Data	Word Embedding	Training Algorithm	Pipelining (if any)	Hyper Parameters (if any)	Score on Test Data/Kaggle Score(If Submitted)
1	Numerical Columns	Count Vectorizer	Logistic Regression	_	_	0.83
2	WebPage Description	Count Vectorizer	Logistic Regression	_	Min_df = 2 (For Count Vectorizer)	0.88.515
3	WebPage Description	Count Vectorizer	SVM with RBF Kernel	PCA (n_components = 1000)	Min_df = 3 (For Count Vectorizer)	0.88815
4	WebPage Description	TF-IDF	Logistic Regression		Min_df = 2 (For Count Vectorizer)	0.85
5	WebPage Description	TF-IDF	Logistic Regression		Min_df = 3 (For Count Vectorizer)	0.86
6	WebPage Description	TF-IDF	SVM with Linear Kernels	PCA	Min_df = 3 (For Count Vectorizer)	0.887
7	WebPage Description	TF-IDF	SVM with RBF Kernels	PCA	Min_df = 3 (For Count Vectorizer)	0.885
8	WebPage Description + Numerical Columns	TF-IDF	SVM with RBF Kernels	PCA	Min_df = 3 (For Count Vectorizer)	0.8885
9	WebPage Description	TF-IDF	Random Forest Classifier		Min_df = 3 (For Count Vectorizer)	0.8532
10	WebPage Description	TF-IDF	XG Boost		Min_df = 3 (For Count Vectorizer)	0.88621
11	WebPage Description	TF-IDF	ADA Boost		Min_df = 3 (For Count Vectorizer)	0.86001
12	WebPage Description	Word2Vec	Random Forest Classifier			
13	WebPage Description	Word2Vec	SVM with RBF Kernel	PCA		

Models Built over html Content

Model Number	Training Data	Word Embedding	Training Algorithm	Pipelining (if any)	Hyper Parameters (if any)	Score on Test Data/Kaggle Score(If Submitted)
1	Html Content	Count Vectorizer	Logistic Regression		Min_df = 10 (For Count Vectorizer)	0.79654
2	Html Content	Count Vectorizer	Logistic Regression		Min_df = 5 (For Count Vectorizer)	0.82564
3	Html Content	Count Vectorizer	SVM with RBF Kernel	PCA (n_components = 1000)	Min_df = 3 (For Count Vectorizer)	0.81722199
4	Html Content	TF-IDF	Logistic Regression		Min_df = 20	0.888806 (Locally, Did not submit)
5	Html Content	TF-IDF	Logistic Regression		Min_df = 10	0.84659
6	Html Content	TF-IDF	SVM with Linear Kernels	PCA (n_components = 1000)	Min_df = 20	0.85731
7	Html Content	TF-IDF	SVM with RBF Kernels	PCA (n_components = 1000)	Min_df = 20	0.8518
8	Html Content	TF-IDF	SVM with PolyNomial Kernels	PCA (n_components = 1000)	Min_df = 20	0.813988
9	Html Content	TF-IDF	Random Forest Classifier	Grid Search CV (to obtain optimal parameters) Max_depth = 20 n_estimators = 50	Min_df = 20	0.87479(Locally, Did not submit)
10	Html Content	TF-IDF	XG Boost		Min_df = 20	0.886974(Localy, Did not Submit)
11	Html Content	TF-IDF	ADA Boost		Min_df = 20	0.8576

Individual Contributions

- To begin with both of us were pretty new to domain of NLP, even the basics were alien to us.
- We started of by learning about the different word embeddings together, until the first evaluation we worked together on EDA and text preprocessing.
- From the 2nd Evaluation onwards:
    - Work By Ghazi: Focussed more on improving the quality of models built using the tf-idf, count vectorizer sparse matrices using the text from webPage Description and the numerical features.
    - Work By Ishan: Focussed more on experimenting with the html_content files, Trained several models over the different word embeddings build over the text from the html content files.
        - This was necessary to determine which resource of text for a webpage is a better feature.
- After Building various models over the text data from webpageDescription, and htmlContent, both of us discussed on finalizing notebooks and compared performance of different word embeddings in different scenarios.
- Overall, it was a great team effort by both of us.

Conclusions

- Here, are some of our insights after extensively with the dataset.
- Comparison of different Word Embeddings
  - Bag Of Words - simply checks the occurrence of a word from the vocabulary.
    - Takes a lot less time to train as compared to something like a skip gram model.
    - CBOW also works pretty well with larger amount of training data, we observed this after we saw that reducing the number of features using the min_df parameter performed significantly better as compared to other models where we did not use min_df in the vectorizer.
  - TF-IDF - takes into account the frequency of a times a word appears in a sample. Thus gives importance to the words, based on the frequency.
    - Training time is roughly the same as BOW. TF-IDF also works significantly better with smaller number of features and a larger vocabulary (as we are now taking account the frequency of a word).
  - Word2Vec - this embedding takes into account the context of the surrounding words. For example, words like “good” and “great” which sound similar, do not occur together in the sparse matrix.
    - We used google’s pre-trained word2vec model to build vect-avgs.
    - In order to train a word2vec model from scratch, we require a lot of training time.
    - Overall what we observed was that the skip grams model didn’t give as good results as compared to tf-idf vector obtained from tf-idf embedding.

Conclusions (Continued …)

- Numerical Features were redundant:
  - Another observation we made during feature selection was that most of the numerical features were redundant, they were not giving really good results when combined with textual features.
- Textual features: Webpage Description better than htmlContent
  - WebpageDescription stemmed strings gave a better result than the stemmed strings obtained from html Content stemmed strings obtained from the html files.
- Number of samples were not sufficient for the given number of features.
  - We also think that the number of training samples were simply not sufficient for the size of the vocabulary that we obtained from text preprocessing (after stemming/lemmatizations).
  - PCA turned out to be extremely helpful, here we were reducing the number of components to be about 1000, this worked pretty well for us as we were able to get really good scores on kaggle simply by training SVMs over data obtained from PCA.
  - 5k samples for about 1k features worked perfectly

- Overall we were able to obtain a decent score of about 0.87 on kaggle. Using this model, we can prett confidently classifying whether a webpage is suitable for posting an Advertisement.

References

- https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b
- https://towardsdatascience.com/text-classification-in-python-dd95d264c802
- https://radimrehurek.com/gensim/models/word2vec.html
- https://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text