Help Boost Our Online Reach

Samaksh Dhingra Shounak Shirodkar

Abstract

Classification task - Exploratory Analysis - Preprocessing - Feature extraction - Choosing the right model(s) - Enhancement and Performance - Conclusion.

1 Problem Statement

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 is 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.
From the given set of information about some webpages, we have to automate process of evaluating it for the suitability of publishing ad on it to get maximum reach and hence mark it 'relevant' or 'irrelevant' based on this.

This is a **Binary Classification** Task.

Raw HTML files are given as data and information provided about a web page also has description for the web page such as title, body content, url, etc., so we have to handle it accordingly to consider it for the evaluation.

This is a **Natural Language Processing** Task.

Evaluation Metric:

Reciever Operating Characterstic-Area Under the Curve (ROC-AUC) Score

2 Technologies

The project was initially conducted in the Colab environment.

We migrated to Kaggle upon encountering CPU limits within Colab during program execution. Kaggle has a higher RAM allotment than Colab, and our program executed without a crash or timeout.

We have made use of the packages Pandas, Numpy, Scikit-learn, nltk, Matplotlib and Seaborn to conduct machine learning tasks such as exploratory data analysis, feature selection, feature engineering and extraction, as well as model training.

BeautifulSoup, a Python package for parsing and scraping HTML documents, was used to extract web content from the html_content folder provided with the dataset.

3 Exploratory Data Analysis

3.1 Overview of Dataset

Dimensions of the dataset:

- Number of Rows (data points) = 5916
- Number of Columns (features) = 27

Out of 27 columns, 2 columns 'url' and 'webpageDescription' contain language data. At this stage, dataset is reasonably "thin". The number of columns are expected to grow upon language preprocessing, therefore the low count of numerical columns now is desirable.

3.1.1 Balanced or Unbalanced?

- Number of '1's in dataset = 3060
- Number of '0's in dataset = 2856

It is evident that the given dataset is fairly balanced. We forego operations such as **undersampling** or **oversampling** intended to balance the dataset and avoid bias in the trained model.

3.1.2 Inferences

Within our numerical data:

- Column 'id' has unique value for every row.
- Column 'framebased' has all its values as 0.
- Columns 'isNews' and 'isFrontPageNews' are categorical.

Within our language data:

- Column 'webpageDescription' contain data in JSON/Boilerplate format
- The *id* tag matches the file name for the corresponding scraped webpage.

3.2 Investigating Data through Visualizations

We performed Exploratory Data Analysis on the numerical features in the data. This was done using **seaborn** library. The intention was to understand the distribution of the numerical data. These observations helped us during Feature Selection.

3.2.1 Correlation Matrix for the data

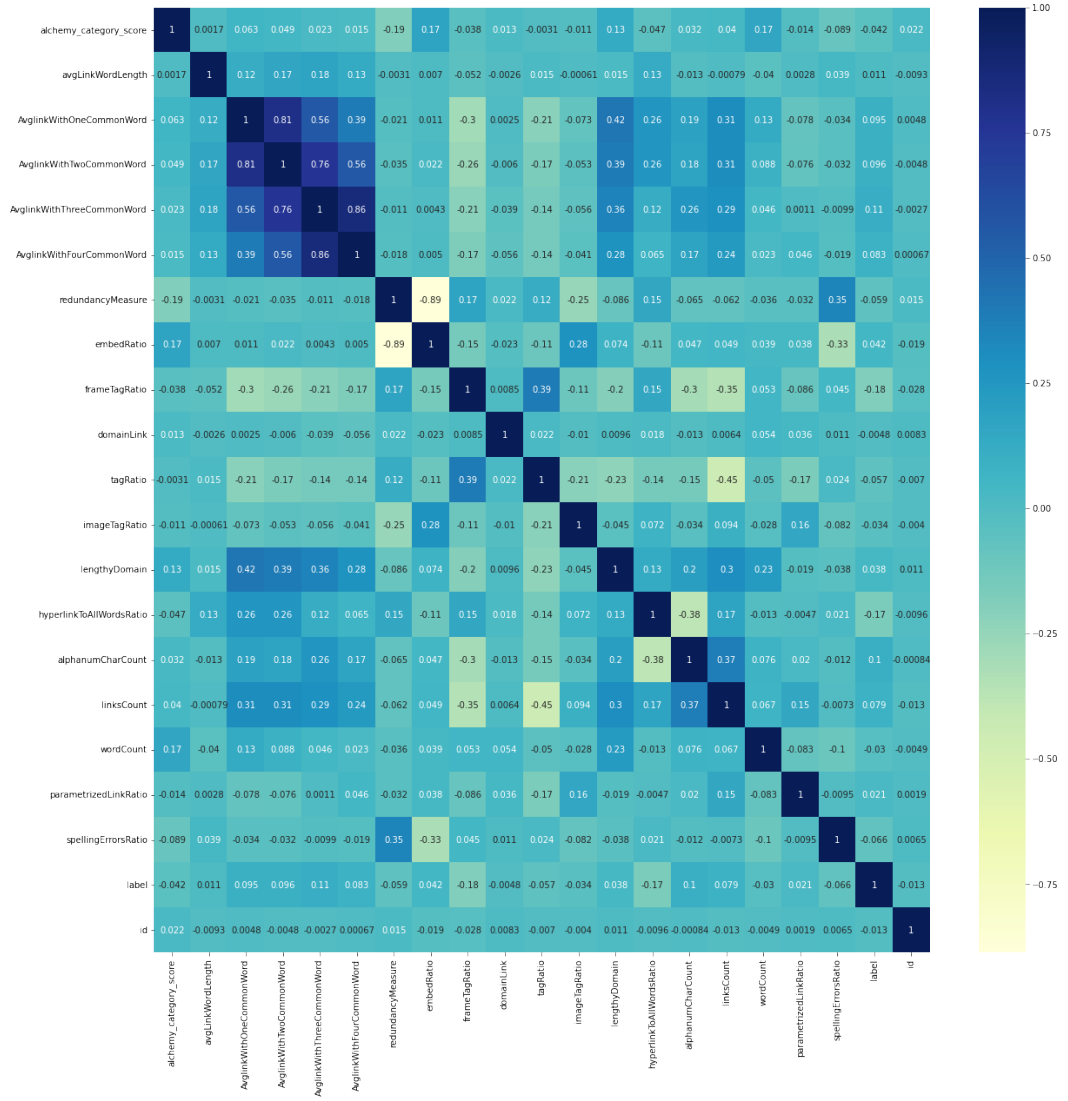

Figure 1: Correlation Heatmap

3.2.2 Skewness in Continuous Data

Histograms were plotted for all continuous features to get a fair idea about skewness in data and its type. Skewness was then handled accordingly in Feature Engineering. It helped in visualising distribution of values in particular numerical column.

Visualizations of skewness for different continuous numerical features:-

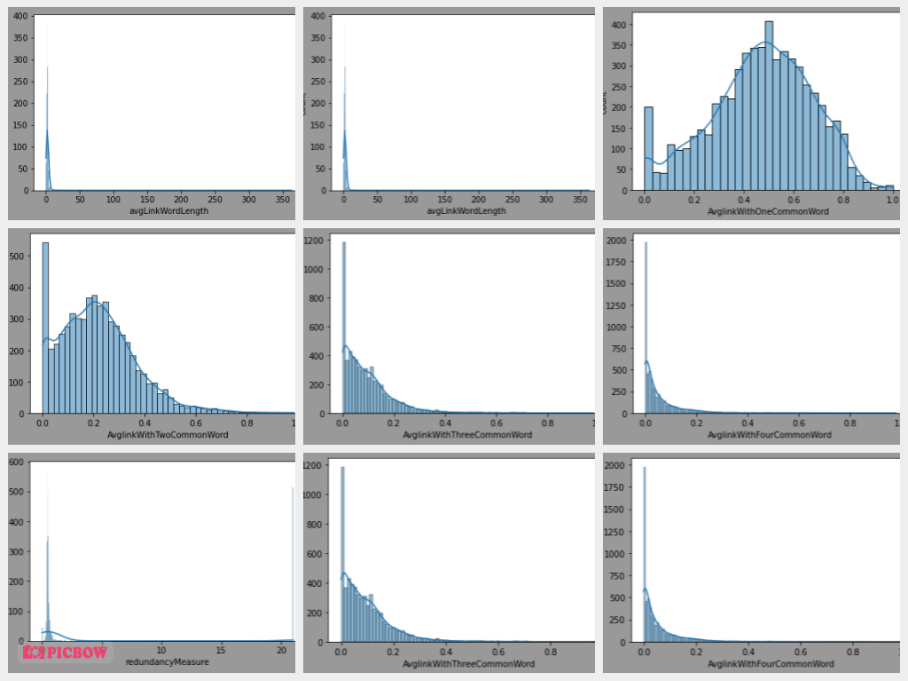

Figure 2: Skewness in Data

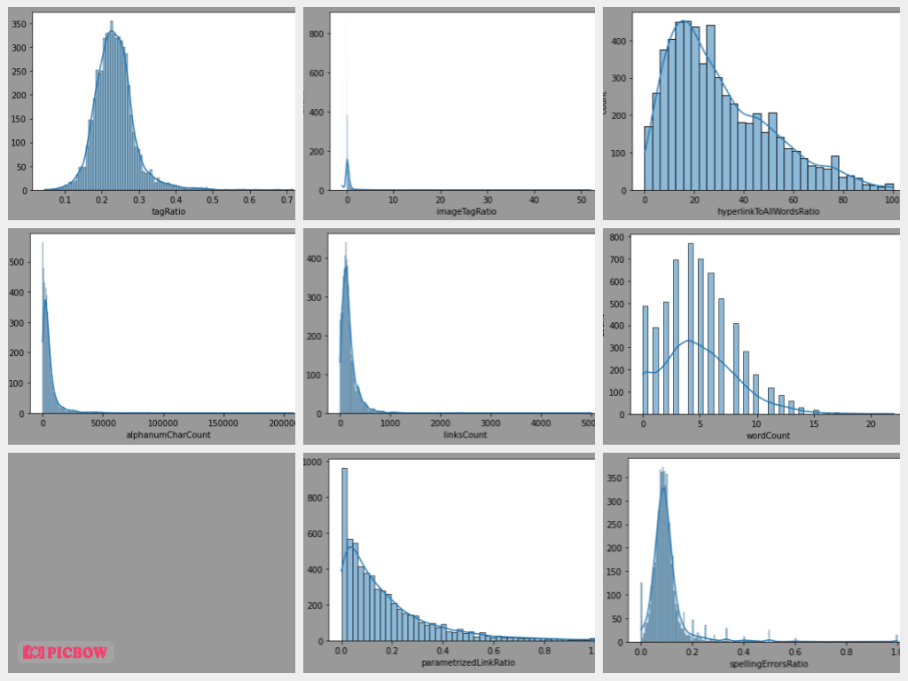

Figure 3: Skewness in Data

3.2.3 Outliers

Boxplots were plotted for all continuous features to visualise the outliers in a numerical column. These were handled accordingly during Feature Engineering.

Visualizations of outliers for different continuous numerical features:-

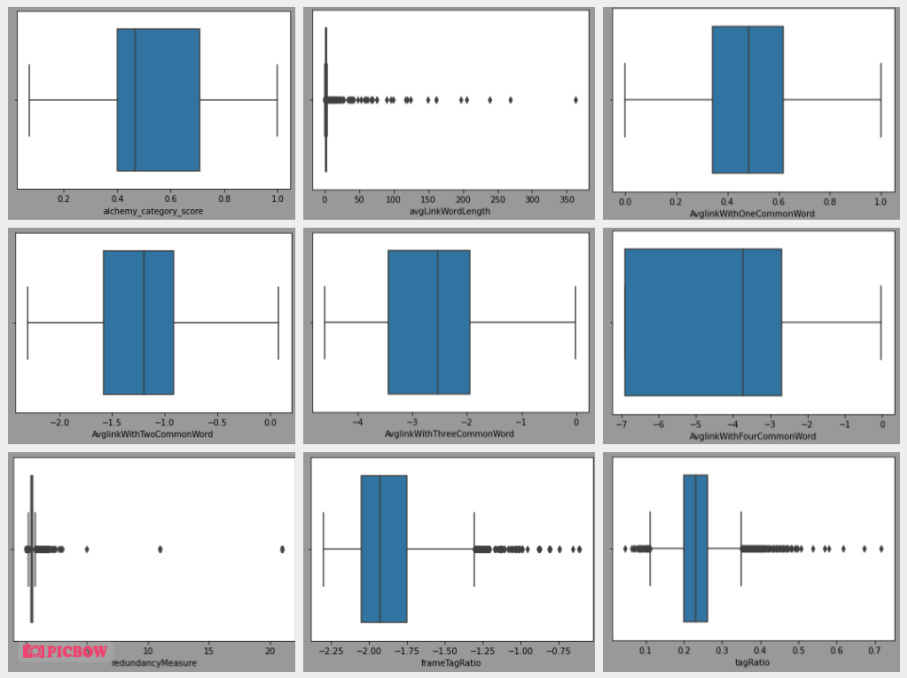

Figure 4: Boxplots to check for outliers in Data

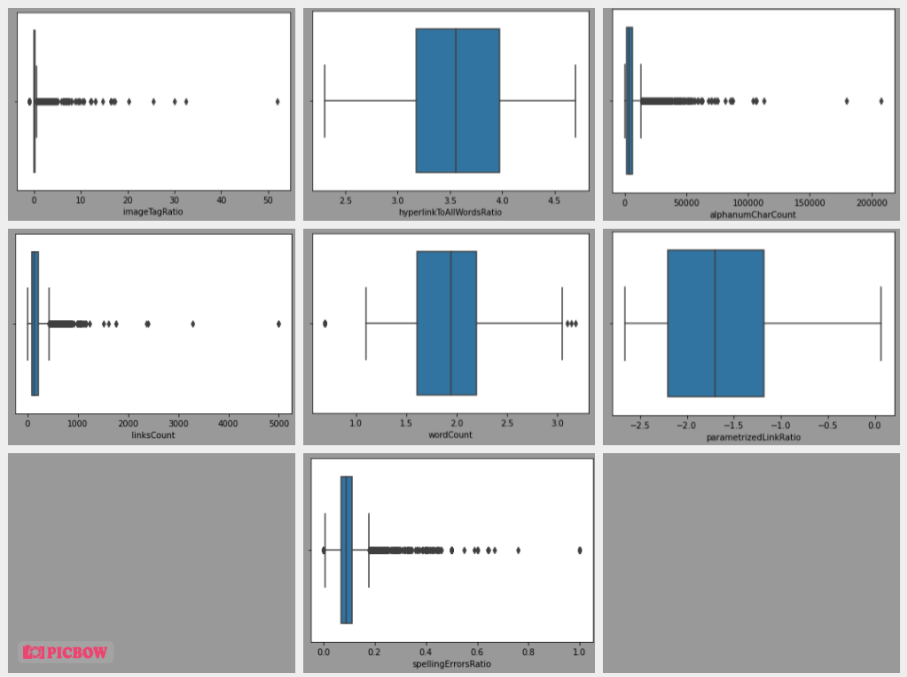

Figure 5: Boxplots to check for outliers in Data

4 Feature Selection & Engineering

The data has to undergo extensive preparation before being used for training within the learning model. Several anomalies and inconsistencies within the data are removed.

The data consists of both numerical and non-numerical components.

4.1 Numerical Data

4.1.1 Redundant Data

Two columns were found to be redundant.

- Column *id* is unique for every every row.
- Column *framebased* has constant distribution.

These two columns were duly removed.

4.1.2 Missing Value Imputation

Several columns were found to have missing values. In each case, proper imputation or complete removal was carried out, keeping in mind the usefulness of data towards model learning.

- *alchemy_category*: All '?' were converted to existing category 'unknown'.
- *alchemy_category_score*: All '?' were assigned values corresponding to the 'unknown' category.
- *isNews*: Dropped entirely due to significant fraction of missing values.
- *isFrontNews*: All '?' replaced with mode value of column, '0'.

4.1.3 Skewness Removal

All skewed distributions were corrected by tranformative removal of skewness through log-transform. These were, namely, *AvgLinkwithTwoCommonWord*, *AvgLinkwithThreeCommonWord*, *AvgLinkwithFourCommonWord*, *frameTagRatio*, *hyperlinkToAllWordsRatio*, *wordCount*, and *parametrizedLinkRatio*.

4.1.4 Outlier Removal

Distributions with outliers, found through boxplot analysis during EDA, were processed by removing these outliers for more uniform training data. These were, namely, *avgLinkWordLength*, *redundancyMeasure*, *frameTagRatio*, *tagRatio*, *imageTagRatio*, *alphanumCharCount*, *linksCount*, *wordCount*, and *spellingErrorsRatio*.

4.1.5 Standardazing Data

Basic Normal Standardization was applied to all the columns of the data.

$$x_{scaled} = \frac{x - mean}{sd}$$

4.2 Language Data

The 'webpageDescription' and content extracted from HTML Files are non-numerical language data, which must undergo language preprocessing into numerical data before being passed as training data to the model.

4.2.1 Textual Data Extraction

1. JSON/Boilerplate Data ('webpageDescription')

   The data was extracted from a JSON string to a Python dictionary object for convenient handling. All columns within this data were added to the json data corpus in concatenation. In this manner, empty tags were compensated by the corresponding non-empty ones.

   'json' library present in python were used to extract the required content easily.

2. HTML File Contents (Raw Scraped HTML Files)

   The HTML data was first extracted from the web content using Beautiful Soup for web scraping. This lead to a more thorough extraction of data as compared to that by Alchemy API, which produced missing values for some of the extracted data.

   Soup object was created using boilsoup method for each HTML file so that we could usefeatures of BeautifulSoup library to extract contents from HTML.

   Following basic steps were performed to clean the html data obtained using BeautifulSoup:-

   - All non textual and unwanted tags were removed from the file.
   - Extracting Text Content of important tags in html - { 'title' , 'h1', 'h2', 'h3' , 'meta-description', 'meta-keywords', 'a' and 'other' }
   - Cleaning the text using 'clean_string' method, which basically lowers the characters, remove $'\backslash b'$, $'\backslash r'$, $'\backslash n'$, etc. All columns within this data were added to the raw data corpus in concatenation. In this manner, empty tags were compensated by the corresponding non-empty ones.

Basic Preprocessing performed on these corpuses were:-

- Removing Punctuation and Numbers- Special characters such as punctuation, numbers, etc. were removed
- Lowercasing all characters- Made all characters in paragraph string lowercase
- Removing Stop-words like 'is', 'has', 'the', etc. were removed as they have little to no relevance in determining result

4.2.2 Word Reduction

Word Reduction technique is mainly to reduce the words to their roots words. This would help in finding more common words so that we could obtain relations easily. This also helps in reducing the number of unique words which will also lead to dimensionality reduction, and better model performance.

We primarily tested two Word Reduction techniques:

- **Stemming**: Stemming reduces inflected words to their corresponding word stems by stripping off the suffix. Stemming helps group similar words and reduces data dimensionality by identifying correlated vectors.

  However, stemming fails in several cases, leading to incorrect word reduction. For example, both *tried* and *try* have the same word root, that is *try*, but a stemming program will reduce the former to *tri*, and the latter to *try*.

- **Lemmatisation**: Lemmatisation groups inflected words with the same word root, or lemma. Unlike stemming, which operates without knowledge of context and simply trims a suspected inflection, a lemmatisation program performs a dictionary lookup to identify the lemma and performs vectorisation accordingly. A dictionary lookup is expensive, therefore lemmatisation is a time-consuming operation.

Words were reduced to their roots by means of lemmatisation. Stemming is a faster and less CPU-intensive method, but sacrifices predictive accuracy. Lemmatisation consumes a lot of RAM and takes time to execute due to the multiple search and compare operations.

4.2.3 Vectorisation

This data finally needs to be vectorised to numerical data. We have tried two main techniques for vectorisation:-

- **Bag Of Words model**: BOW, also known as Count Vectoriser, is a simple vectorisation technique, that simply keeps a tally of frequency, disregarding grammar, word order and relation. Naturally, BOW will lead to a less accurate model.

- **Term Frequency-Inverse Document Frequency**: TF-IDF uses a statistic that describes the specificity of a word as inversely proportional to the number of different documents it appears in. For this reason, adjustments are made for words that appear more frequently in general such as stopwords.

  TF-IDF vectorisation was chosen for its relevance to this particular problem statement.

  TF-IDF measures the importance of a word in a document within a corpus, is more relevant in this context. The more importance the website will have, the higher it will figure in search rankings. Top search results typically have much higher click-through rates, and are more viable for ad display.

  Most notably, TF-IDF was used within Google's search engine until replacement with the more advanced BERT technique.

- **Word2Vec**: Word2Vec is a more recent vectorising technique that utilises a shallow neural network under the hood to learn word associations and produce word embeddings through a measure of semantic similarity (typically, cosine similarity).

  W2Vec allows two architectures, CBOW and Skip Gram. Here, we gave preference to CBOW noting that it is less computationally expensive while providing similar results.

  Note that there is no restriction to exclude the test set text data from the corpus. W2Vec does not perform any predictive behaviour with respect to semantic similarity, and it is beneficial to have a large vocabulary and deep knowledge of word associations.

  We have not considered multi-word ngrams, which would in theory improve performance, because common text makes use of phrases and idioms, which are ignored here.

4.3 Dimensionality Reduction

The number of columns balloon to a large value after language preprocessing, given that our collection of webpages (and hence, our corpus), is diverse and extensive.

To optimise model performance, as well as improve model accuracy, dimensionality reduction must be carried out over the transformed data to obtain less correlated and lower volume data.

Although, techniques like Stopwords Removal, Punctuation Removal, Word Reduction (Stemming or Lemmatization) do help in reducing dimensionality of the vector obtained after vectorisation by removing unnecessary words from the corpus that have almost very little to no effect on the output.

We used some following formal techniques to significantly reduce the dimensions of the vector obtained:-

- **Principle Component Analysis**: PCA involves reduction of a dataset with highly correlated features to an equivalent dataset based on identification of principal components by projecting into a lower dimensional subspace via an eigendecomposition.
- **Single Valued Decomposition**: SVD, a mathematical construct from linear algebra, allows us to decompose a large matrix with correlated columns into a sum of outer products with lesser correlation.

Sklearn provides built-in functions for these.

5 Understanding Loss Metric

This being a binary classification task, our natural choice for a loss metric was the ROC-AUC score.

The Receiver Operating Characteristic curve (ROC) is a graphical plot between the true positive rate (TPR) and false positive rate (FPR) that illustrates the diagnostic ability of a binary classifier.

ROC curves are a powerful statistical measure of classification performance, due to their precise nature, and are widely used.

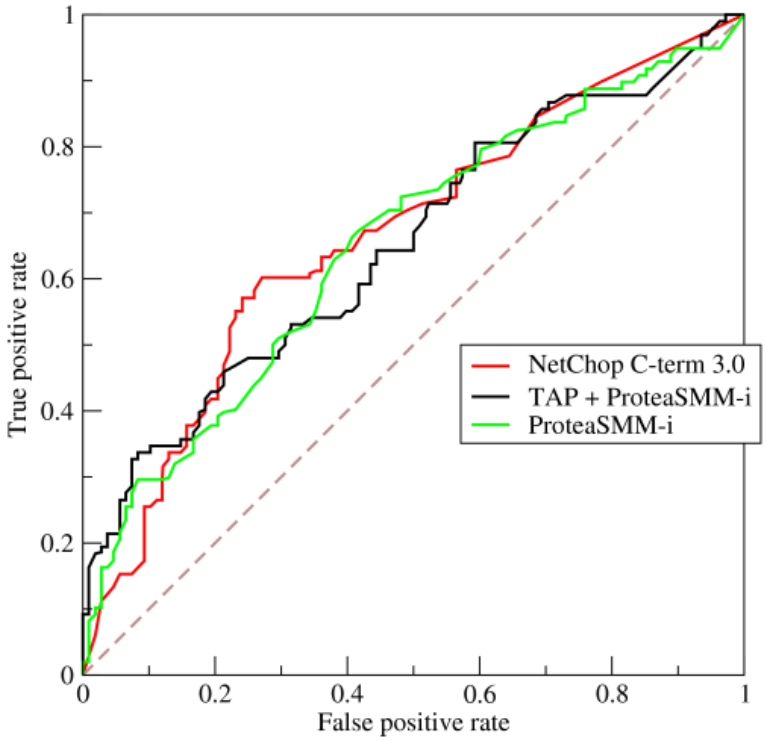

Figure 6: The ROC curve (credit: Wikimedia)

TPR may be defined as the probability of detection, while FPR is the probability of false alarm.

The ideal classifier has perfect separability and AUC for its ROC equal to 1. Similarly, the worst classifier misclassifies every data point, and has its AUC equal to 0. The perfect non-classifier is unable to distinguish between classes and has AUC equal to 0.5.

Intuitively, one may picture how the ROC-AUC might measure model quality.

6 Model Selection

The goal is to select the best performing and most suitable model from our set of candidates permissible for this competition. Our primary criteria for model selection was performance during cross validation.

Cross validation is the most accurate, yet most computationally expensive, criterion for supervised model selection. We used five-fold cross validation to compare our models in case of disputes.

We gradually progressed from simple to complex model constructions, in the hope to first discover the simplest possible optimal model for our problem, as suggested by Occam's Razor principle.

Our rationale behind every model as well as their working is briefly introduced.

6.1 Naive Bayes

6.1.1 Multinomial Naive Bayes Classifier

MNB is a probabilistic learning algorithm very common to natural language processing. The model operates under the *naive* assumption that all the features are mutually independent.

The MNB Classifier is suited for classification with discrete features (especially integers), however it is observed to give good performance even with fractional values, such as TF-IDF.

Our model was trained only upon vectorised language data, and yielded a high accuracy score of around 0.86, despite having ignored the numerical component of our data completely.

6.2 Linear models

Linear models are the simplest learners, learning parameters for its linear transformation towards optimal classification.

6.2.1 Logistic Regression Classifier

The learner is a generalisation of simple logistic regression to multi-class classification. The underlying assumption is that our data have low collinearity to provide some separability.

6.2.2 Stochastic Gradient Descent Classifier

The learner is comprised of a linear model with a regularisation penalty with application of stochastic gradient descent learning. The gradient of the loss for each sample is updated at every step as per a preset learning rate. Regularisation helps avoid high variance in the resultant model, common with gradient descent optimisation.

6.2.3 SVM Classifier

The Support Vector Machine is a non-probabilistic classifier in our context. The basic idea behind model learning is to maximise width of the gap between two classes and their separator. To negotiate non-linear classification boundaries, a transform to higher dimensions, called the kernel trick, is used.

We primarily used the linear and the RBF (Radial Basis Function) kernel in our experiments.

6.3 Ensembles

The ensemble method utilises multiple learners to obtain better predictive performance than any one of the learners individually might yield. The intention is to obtain multiple good hypotheses and combine their resultant to maximise predictive accuracy.

The two chief ensemble methods we tested were Bagging and Boosting.

6.3.1 Bagging

Bootstrap Aggregating, also known as Bagging, is an ensemble method.

Random Forest is a Bagging ensemble. It uses majority vote over a number of decision trees trained over random samples of the training data. This is purposed to avoid overfitting to training data and promoting model generalisation.

However the model is prone to be high bias.

6.3.2 Boosting

Boosting is an ensemble method utilising a group of weak learners to output a strong learner.

Decision tree boosting is a decision tree ensemble that combines weak decision trees whose performance is only marginally better than random classification, to produce a strong classifier. The most popular tree ensembles utilise gradient boosting.

Gradient boosting typically outperforms Random Forest models. Also, gradient boosted models are prone to high variance, due to their tendency to overfit training data.

7 Experimentation and Challenges Faced

As opposed to theoretical expectations in the previous section, this section details our practical observations.

7.1 Initial Stage

We tried different preprocessing techniques for a basic model.

1. We performed various feature extraction and selection techniques at this stage
2. We tried including different combinations of non textual data with vectors of data with help of Exploratory Data Analysis
3. We observed that when we used only vectorized natural language data in 'webpageDescription' and excluded all numerical data gave the best performance for the basic logistic regression model.
4. Word reduction didn't improve the results.

7.2 Intermediate Stage

At this stage, we explored various model with different hyperparameters to find the best fit for our data. The best model was chosen through extensive experimentation. Our attempts are detailed in order:-

1. Our first model, a simple multinomial Naive Bayes classifier, was trained using only the natural language data. Enhancements in accuracy were achieved through TF-IDF Vectorization during preprocessing.
2. The simple logistic regression classifier, with log-likelihood as the loss metric, also yielded a good performance over the training data.
3. The SGD Classifier (linear classifier using stochastic gradient descent to optimise parameters) performed very well, over both training and test data.
4. The calibrated SGD Classifier performed even better, utilising its built-in isotonic hyperparameter tuning.
5. The ensemble methods: random forest (bagging) and boosting (XGBoost), both, did not perform better than our test with SGD Classifier (Performed after Principle Component Analysis on Vectorized Data)

7.3 Final Stage

There was a problem running these on Colab due to the CPU limits, however this was circumvented by utilising Kaggle. Here, we noticed that random forest overfit the training data and XG Boost also did not give any better results.

With Kaggle, we directly trained our bagging-boosting and svm models on our data without dimensionality reduction.

1. SVM Classifier did not improve over our previous record accuracy. The SVC with the default RBF kernel took long to execute, so that there was a session timeout within Colab, considering our large dataset. Note that this was prior to dimensionality reduction, especially as the number of columns is large from the vectorised natural language component of the data. The linear kernel did lead to a performance improvement, but there was no improvement in terms of accuracy.

   We tried the RBF kernel as well upon shifting to Kaggle. But both of these did not yield a satisfactory performance as compared to Logistic Regression and Multinomial Naive Bayes, and the update was rolled back.

2. We included the extra information from the given raw html files at this stage and used Beautiful Soup Library to do it. We then performed basic cleaning on the text data from the raw html files.

3. Training the model only on the corpus extracted from raw HTML data (using BeautifulSoup) gave ROC-AUC score of 0.5.

4. Using the *webPageDescription* corpus without adding this extracted data gave a better ROC-AUC score.

5. For final chance, we decided to perform bit more cleaning on Raw HTML text data and JSON text data with help of lemmatization and stopwords removal. We formed seperate corpus for both text data and vectorized using RF-IDF Vectorizer. Since We had 2 large dimensions of vectorized arrays, memory had reached it's peak, so we decided to perform Singular Valued Decomposition (SVD) on it seperately with 120 components. We then trained 2 Logistic Regression Models with 'liblinear' solver on them respectively. We created an ensemble out of these 2 models keeping weight of JSON one twice as other one. Unfortunately, the results were not as expected.

Challenges Faced:-

- Memory Constraints: We could not try many things like word2vec initially due to memory constraints in the server.
- Timed Sessions: Every session has a timeout duration on online notebooks. For longer execution times, we would often find that our session would disconnect while the model was training from hours.
- Cleaning the Raw HTML Files: Due to lack of experience, we found it difficult to perform extensive or customised cleaning on raw html files.

8 Final Model(s) Used

For Notebook 1 (Best Score)

Our final model utilises an ensemble comprised of a Logistic Regression model (using the liblinear implementation) and a Multinomial Bayes Classifier (with alpha=0.45). The ensemble was trained using logistic regression. The optimal result was obtained with a 1:2 combination of results from Multinomial Naive Bayes Model and Logistic Regression model respectively.

Experimentation with the ensemble was a logical next step after our realisation that both these models individually gave ROC-AUC scores around 0.87.

For Notebook 2 (Submission with most efforts and expectations)

The model utilizes an ensemble comprised of 2 Logistic Regression Models (using liblinear implementation). First one trained on Json Text Data and other one trained on Raw Html content text data. The result is obtained with 1:2 combination of raw html one and json one respectively.

9 Table of Models and their Scores

S no.	Model Used	Hyperparameters	Basic Preprocessing	Private Score	Public Score
1	Logistic Regression	Default	Missing Values Handled- No NLP and EDA	0.66526	0.67183
2	Multinomial Naive Bayes	Alpha = 0.45, others default	Only NLP on Body, special char removal, stemming, bag of words with 1000 limit	0.84379	0.81622
3	Multinomial Naive Bayes	Alpha = 0.45, others default	Same as previous, bag of words with 10000 limit	0.85473	0.82244
4	Multinomial Naive Bayes	Alpha = 0.45, others default	Same as previous, bag of words with no limit	0.85961	0.82725
5	Multinomial Naive Bayes	Alpha = 0.45, others default	Same as previous, Used Lemmatization instead of Bag of words	0.87586	0.86229
6	Multinomial Naive Bayes	Alpha = 0.45, others default	Same as previous, Included Title and Url content also with body in corpus	**0.88491**	**0.87392**
7	Stochastic Gradient Descent Classifier	Loss = 'log', penalty = 'l2', alpha = 1e-3, max_iter = 5	Same as previous	0.86924	0.88473
8	Stochastic Gradient Descent Classifier	Loss = 'log', penalty = 'l2', alpha = 1e-3, max_iter = 5	Same as previous, added raw text instead of json text content in corpus	0.8692	0.88485
9	Calibrated Classifier CV	base_estimator = prev SGD Classifer, cv = 3, method = 'isotonic'	Same as previous	0.86793	0.88623
10	Support Vector Classifier	Kernel = 'linear'	TF-IDF(5000), PCA(500), Same as previous	0.83324	0.86233
11	Support Vector Classifier	Kernel = 'rbf', C= 0.1	Same as previous	0.85642	0.85229
12	Support Vector Classifier	Kernel = 'rbf', C= 0.1	No Limit on TF-IDF, No PCA, Same as previous	0.86961	0.87792
13	Random Forest Classifier	n_estimators = 500	Same as previous	0.79275	0.80296
14	XG Boost Classifier	Default	Same as previous	0.85221	0.87039
15	Logistic Regression	Default	Concatenated json extracted text instead of raw data,Same as previous	0.8709	0.88574
16	Logistic Regression	Solver = 'liblinear'	Same as previous, No Lemmatization	0.87093	0.88706
17 (Notebook1)	**Ensemble((2*Logistic Regression + Multinomial NB)/3)**	**LogisticReg(solver = 'liblinear'), MultiNB(alpha = 0.45)**	**Same as previous**	**0.87364**	**0.89043**
18	Ensemble((Logistic Regression + Multinomial NB)/2)	LogisticReg(solver = 'liblinear'), MultiNB(alpha = 0.45)	Same as previous	0.87338	0.88984
19	Logistic Regression	Default	Only Text Extracted from HTML Contents, Tf-idf	0.50423	0.49406
20	Logistic Regression	Solver = 'liblinear'	Same as previous, also added Json extracted text in corpus	0.82074	0.79282
21 (Notebook2)	Ensemble((2 Logistic Regression)	Solver = 'liblinear'	Corpus 1 made by json extracted text data with lemmatization, Corpus 2 made by raw html text data with lemmatization, Both tf-idf vectorized, SVD(120)	0.80196	0.8158

Figure 7: Scores Table

10 Contributions

From previous experience, we felt it makes sense for one person to have control over the code structure, because collaborations are slow, and we could not afford a delay, especially as we had tight deadlines to meet. This notebook was shared between us for edits and help in debugging.

We shared collective responsibility for the project, and the lines between our contributions are blurred. However, here is an attempt to demarcate our contributions.

Samaksh Dhingra (IMT2019075)

1. Maintaining the Python notebooks containing all our programs.
2. All Preprocessing on Numerical As well as Text Data (except Word2Vec)
3. Discussing, Thinking, Studying and Research to improve scores and performance
4. Testing different methods to improve score and making kaggle submissions
5. Contributed in slides for Intermediate and final Evaluations
6. Presented for First and Third Evaluation
7. Suggesting and handling migration to Kaggle.
8. The HTML data extraction using BeautifulSoup parser library.
9. Trying ensembles and various models for phase three.
10. Contribution in writing Final Report

Shounak Shirodkar (IMT2019083)

1. The first round of EDA.
2. Contribution of code snippets.
3. Studying and evaluating language processing techniques for relevance to our project; also implementing Word2Vec.
4. Presented for second evaluation.
5. Contributed in slides for intermediate evaluations.
6. Scouting models and ensembles during phase one and two, through extensive reading.
7. Documenting all plans and attempts in report.
8. Contribution in writing Final Report

11 Conclusion

We managed to fulfill nearly all our proposed targets for this project. In doing so, we studied a bit about NLP and different machine learning models, to build and improve our model.

Our optimal score is around 0.87, and our model predicts ad-worthy pages reasonably well.

We observed a marked difference between our expectations from a theoretical perspective and our observations during practical experimentation with different combinations of models and feature engineering techniques.

Unexpectedly, our best scores came from completely neglecting the numerical component of our data, and only regarding the natural language component.

We had to discard some of our best ideas because the model lost on predictive performance with the update. For example, we expected better accuracy after SVD for reducing dimensionality of vectorised natural language data, but this did not happen significantly, as expected.

We also observed that boosting models are not suited for our data. Boosting always overfit on our training data but performed badly for test data. Intuitively, this might be because of the very nature of our data.

The naive Bayes and logistic regression ensemble performed fairly well with our natural language data as expected theoretically.

12 References

1. The Analytics Vidhya and Towards Data Science publications on Medium for a practical introduction to natural language processing, as well as several learning models.
2. The Stackexchange sites, especially the Data Science forum, for several clarifications as well as conceptual explanations from subject experts in the community.
3. The Numpy Documentation during programming tasks, and to resolve bugs.
4. The Pandas Documentation to resolve recurring problems during implementation.
5. The Scikit-learn website for neat documentation and usage details.
6. The Machine Learning Mastery website for its simple explanations that were in demand throughout this project.