Final Report

Help Boost Our Online Reach

ML PROJECT

TEAM - WeSmurfing
IMT2019073 Ronit Jain
IMT2019516 Rakshit Gupta

Problem Statement

The task is to identify the relevant, high-quality web pages from a pool of user-curated web pages to identify “**ad-worthy**” web pages. The challenge required 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. To facilitate this, the agency has created a dataset of raw Html, meta statistics, and a binary label for each webpage. The binary label represents whether the webpage was selected for ad placement or not.

About the Dataset

The data set was provided on Kaggle. The data set included three files:

- train.csv - the **training set** containing the target variable **'label**.'
- test.csv - the **test set**, which does not contain the target variable 'label.'
- html_content.zip - A zip file containing the **raw HTML content** for each URL

The dataset contains an **ID column**, **25 features** and **one target *label* column.** This dataset includes rows of web pages and their description and meta-statistics, along with a label classifying them as either "**ad-worthy**" or **not "ad-worthy"**, in **binary values 0 and 1**, respectively.

The **training data** set contained **5916** data points.
The **test data** set contains **1479** data points.

Exploratory Data Analysis

Countplot

Displayed the number of
Label counts.

Outliers

Displayed boxplot to find
outliers in the data.

Correlation Map

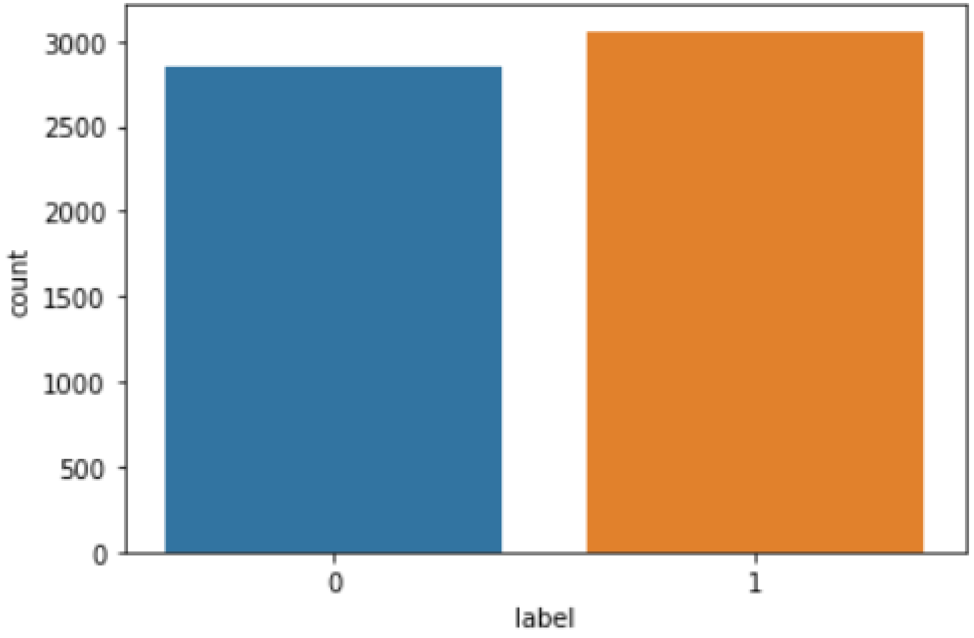
Created heatmap for the rest
numerical features.

Countplot Label Counts

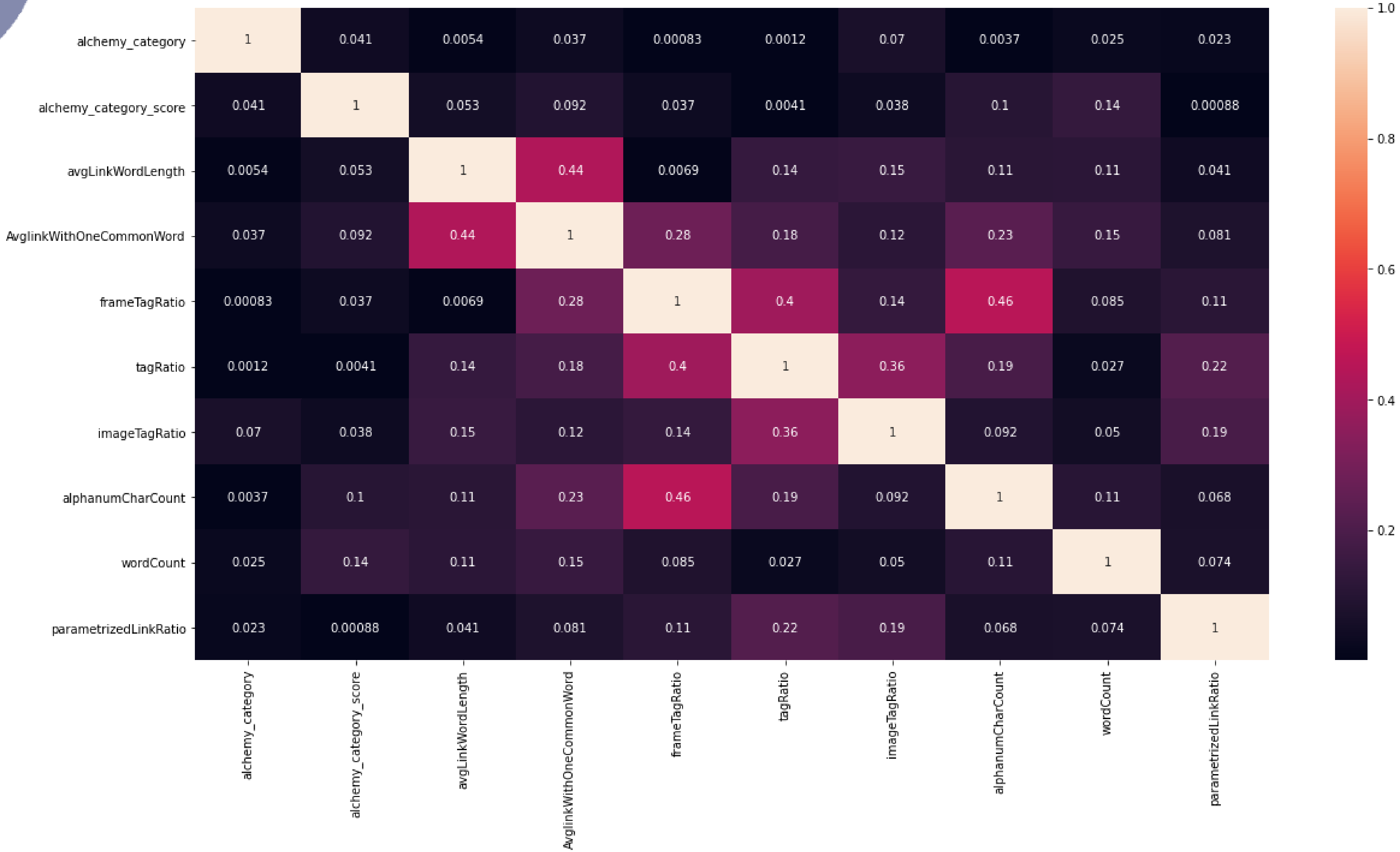
```
print(df_train["label"].value_counts())
sns.countplot(x='label',data=df_train)
```

[246] 0.1s

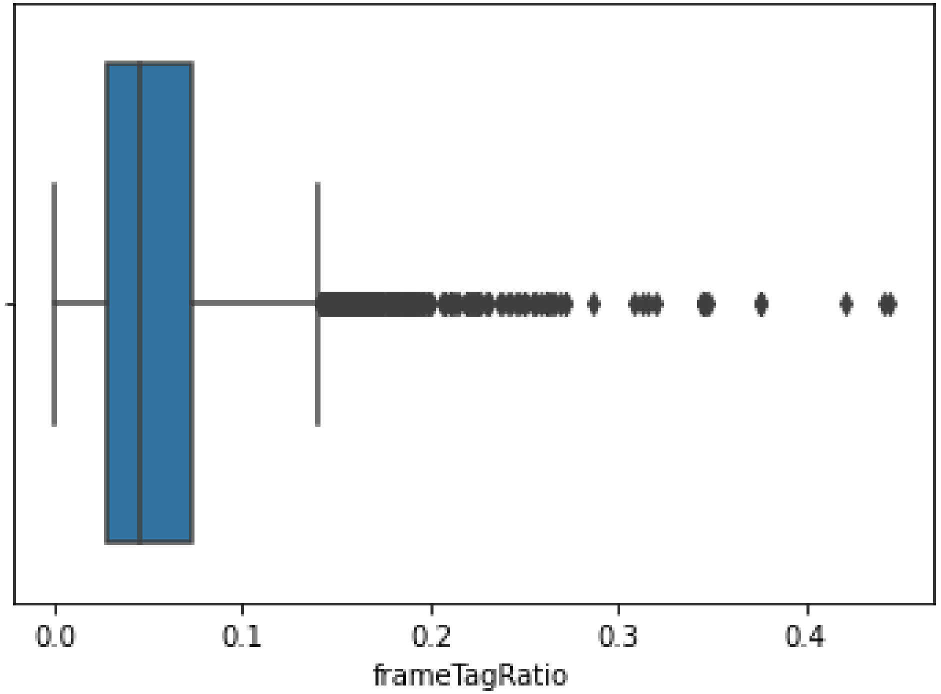
```
1    3060
0    2856
Name: label, dtype: int64

<AxesSubplot:xlabel='label', ylabel='count'>
```

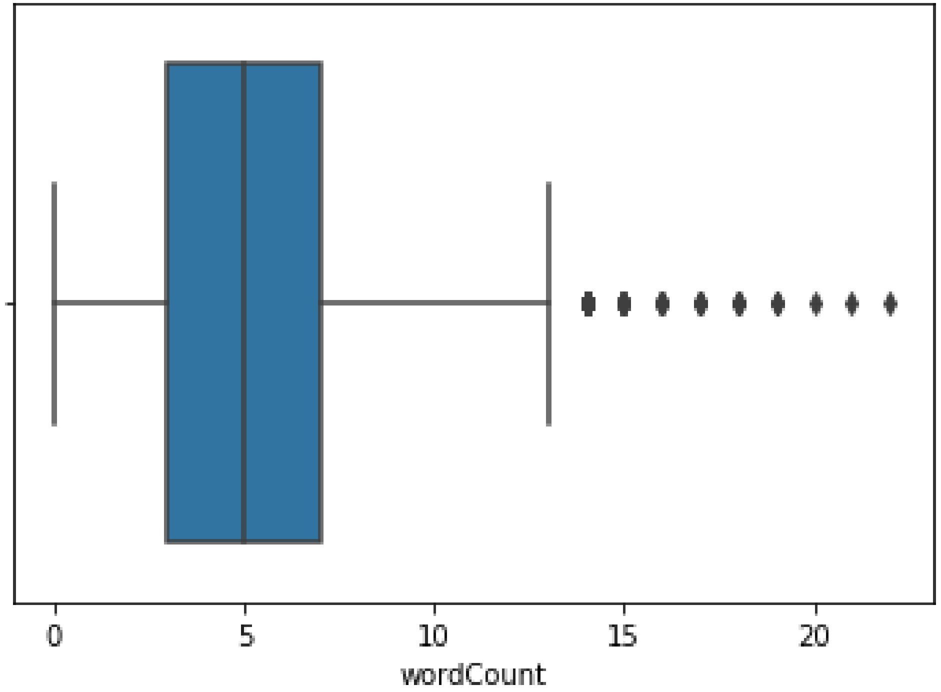


Correlation Heatmap



Outliers





… also for many more features

Preprocessing

- The feature "**webpageDescription**" has lots of data in text format, which will eventually be converted into numerical data.
- The feature named "**url**" is **dropped** because it's of no use in predictions.
- Removed the features "**framebased**" and "**isNews**" because they have only one kind of data.
- The **missing data** is presented in the next slide for both **train** and **test data** combined.

Missing Data

	Missing_Data	Percentage
isNews	2843	38.444895
alchemy_category	2342	31.670047
alchemy_category_score	2342	31.670047
isFrontPageNews	1248	16.876268

- All **NaN** values are replaced with "**unknown**" in "**alchemy_category**", **median** in "**alchemy_category_score**", and **mode** in "**isFrontPageNews**".

Outliers Removal

- We used the following method to deal with the outliers.
- In the following function, if there's an outlier on either side of the whiskers, then that point is reset at the nearest whisker point.

```
class OutlierRemoval:
    def __init__(self, lower_quartile, upper_quartile):
        self.lower_whisker = lower_quartile - 1.5*(upper_quartile - lower_quartile)
        self.upper_whisker = upper_quartile + 1.5*(upper_quartile - lower_quartile)
    def removeOutlier(self, x):
        return (x if x <= self.upper_whisker and x >= self.lower_whisker else (self.lower_whisker if x < self.lower_whisker else (self.upper_whisker)))
```

```
for i in range(1, 21):
    column = df.iloc[:,i]
    # print(column.name)
    column_outlier_class = OutlierRemoval(column.quantile(0.25), column.quantile(0.75))
    outlier_removed_column = column.apply(column_outlier_class.removeOutlier)
    df[column.name] = outlier_removed_column
```

Feature Selection and Engineering

- **After** dealing with **outliers**, there were some features with the **same** kind of **data**, like "**embedRatio**", "**domainLink**", and "**isFrontPageNews**", so we **dropped** them.
- We used **correlation** and plotted **heatmap** to fetch out the **valuable features**. Also, to remove the **highly correlated** features.
- We also used **PCA** for **feature reduction**, but it didn't give better results, so we **dropped** it.
- We used the **vectorization** method to **create features** for the text data in "**webpageDescription**".

NLP

```
def nlp_func(text):
    text = "".join([char for char in text if char not in string.punctuation])
    tokens = re.split('\W+', text)
    # text = [ps.stem(word) for word in tokens if word not in stopword]
    text = [wn.lemmatize(word) for word in tokens if word not in stopword]
    return text
```

- We used the above methods like **removing punctuation**, **tokenization**, **removing stopwords** and **lemmatization** on the text data of the feature "**webpageDescription**".
- We also tried it using **stemming** instead of **lemmatization** but didn't get better results.

Vectorization

- We used the **TF-IDF Vectorizer** to convert the text into numeric data and **create features**.
- This process rules out more than **1lakh features**, so we used some **hyperparameters** as shown below to **limit** the no. of **features** to < **50k**.
- We used the above-discussed **nlp_func** for the **analyzer** part.

```
tfidf_vect = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', tokenizer=lambda x: re.findall(r'[^\p{p}\W+', x), analyzer=nlp_func,
                             token_pattern='(?u)\\b\\w\\w+\\b\\w{,1}',min_df=5, norm='l2', ngram_range=(1,2))

x_tfidf = tfidf_vect.fit_transform(df['webpageDescription']).toarray()
x_tfidf.shape
```

- We used "**StandardScaler**" to scale the rest **10 features** of the dataset and then **merged** them with the **features** received **after vectorization**.

Models Used

1. Logistic Regression Model

```
LR_model = LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None,
                                solver='lbfgs', max_iter=10000, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
LR_model.fit(train_X, train_y)
```

2. Random Forest Model

```
RF_model = RandomForestClassifier(n_estimators=1000, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
                                    max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None,
                                        random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
RF_model.fit(train_X, train_y)
```

3. Naive Bayes Model

```
NB_model = GaussianNB()
NB_model.fit(train_X, train_y)
```

Scores
(Area under the Curve)

Logistic Regression Model	0.8820
Random Forest Model	0.8691
Naive Bayes Model	0.7328

Conclusion

- This project helped us gently get the idea of the domain of NLP.
- We used it to convert the string attributes of the dataset to some form of numerical data and then constructed ML models on this numerical data.
- We also faced some challenges; eventually, those problems only helped us learn new things and apply them.

Individual Contributions and References

Individual Contributions

We did the whole project as a team. We discussed everything and implemented everything together.

References

https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b

https://towardsdatascience.com/text-classification-in-python-dd95d264c802