Project: Help Boost Our Online Reach!

Project Report

By:

Jasvin James Manjaly (MT2021059)

Khushal Abrol (MT2021063)

Problem Statement

Consider yourself to be a consultant for an online advertising agency. The agency spends a considerable amount of time and money to find the best web pages to publish their ads on. They select web pages that will generate prolonged online traffic so that their ads can have a long-lasting reach.

Now, wouldn't it be great if you could somehow automate this process and save company resources? 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.

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 you 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. This task aims to gently introduce you to the domain of NLP, as you would be required to convert the string attributes of the dataset to some form of numerical data, and then construct your ML models on this numerical data.

Can this fast-paced ad agency bank on you to deliver on this project?

Note: Prediction based on probability scores and not on class labels directly!

EDA: Preprocessing + Analysis

Webpage Description column

- The webpageDescription column consists of JSON data which can contain three keys in total, "body", "title", "url".
- Some rows have all 3 keys, some have only 1 and so on.
- We assumed that the "body" key would contain the one-line description of the webpage as described in the problem statement.
- There were 57 entries where the webpageDescription had no "body" key and there were many more instances where the JSON had a "body" key but its value was set to None, such entries are unusable as well.
- So the webpageDescription column was filled in using the following logic,

```
if "body" in webpageDescription and webpageDescription["body"] != None
       return webpageDescription["body"]

else if "title" in webpageDescription and webpageDescription["title"] != None
      return webpageDescription["title"]
```

```
else
      return webpageDescription["url"]
```

There was only one entry that had neither “body” nor “title” nor “url” key, so that entry was dropped.

Analyzing the News columns

```
cleaned_data['isNews'].value_counts()
```

```
1    4552
?    2842
Name: isNews, dtype: int64
```

```
cleaned_data['isFrontPageNews'].value_counts()
```

```
0    5853
?    1247
1     294
Name: isFrontPageNews, dtype: int64
```

- We found a page with the following content in webpageDescription that had isNews marked as ‘?’

  'In a discovery that has stunned even those behind it scientists at a Toronto hospital say they have proof the body's nervous system helps trigger diabetes opening the door to a potential near cure of the disease that affects millions of Canadians Diabetic mice became healthy virtually overnight after researchers injected a substance to counteract the effect of malfunctioning pain neurons in the pancreas I couldn t believe it said Dr Michael Salter a pain expert at the Hospital for Sick Children and one of the scientists Mice with diabetes suddenly didn t have diabetes any more The researchers caution they have yet to confirm their findings in people but say they expect results from human studies within a year or so Any treatment that may emerge to help at least some patients would likely be years away from hitting the market…’

- It is clear that this is a news article and its 'isNews' value was set to '?'
- This just tells us that we'll have to closely analyze each entry with '?' for isNews to determine whether they're a news page or not.
- Or we can consider this to be an outlier and assume that its mostly non-news page whenever isNews = '?'
- A separate NLP model can be trained to predict whether that page isNews or not given the description of the page. But this can be very unreliable plus we don't have class label 0 entries for isNews in the given dataset. So it makes more sense to straight up manually label each entry for whether it is news or not.

- Similar manual labelling can be done for isFrontPageNews as well
- Whichever entry had isFrontPageNews == ?, the same entry also had isNews == ?

Alchemy Category column

```
cleaned_data['alchemy_category'].value_counts()
```

```
?                     2341
recreation            1229
arts_entertainment     941
business               880
health                 506
sports                 380
culture_politics       343
computer_internet      296
science_technology     289
gaming                  76
religion                72
law_crime               31
unknown                  6
weather                  4
Name: alchemy_category, dtype: int64
```

So a majority of the entries have ? values.

```
cleaned_data[cleaned_data['alchemy_category'] == 'law_crime'].head()
```

	url	webpageDescription	alchemy_category	alchemy_category_score
66	http://vii2012.com/2012/06/accidental-90s-nick...	Your smiling at me is my daily dose of magic S...	law_crime	0.505332
377	http://www.independent.co.uk/news/world/americ...	google ad client ca pub 5964551156905038 if re...	law_crime	0.818603
639	http://newsfeed.time.com/2012/01/09/reading-wh...	Monday s links talk surprising birthdays body ...	law_crime	0.752151
847	http://www.insidethekaganoffkitchen.com/2010/0...	by Rachel on April 11 2010 I just love a good ...	law_crime	0.246582
1334	http://www.magnoliarouge.com/2012/08/outfit-of...	AboutWelcome to Magnolia Rouge a collection of...	law_crime	0.313761

This tells you how unreliable the alchemy_category actually is because quite a few of these pages are completely unrelated to the “law_crime” category that is being labelled on them.

The AlchemyAPI website given in the problem statement links to a dead page, so nothing further can be understood about how these categories were determined. A logical assumption is that AlchemyAPI tags a website into different categories on the basis of their HTML content and

the confidence of the tagging is represented by the alchemy_category_score column. So lower the score, lower is the confidence.

This also adds more problems on the validity/accuracy of the data given to us. Because quite often the alchemy_category is misclassified.

We thresholded the score at 0.6 and this gave 2733 rows which means that there are in total 2733 rows where we have alchemy_category_score > 0.6, which means atleast a 60% confidence in the fact that the alchemy_category is correct.

In total we have 7394 entries out of which 2341 are ? entries and 2733 entries with high confidence, so we have 7394 - 2341 - 2733 = 2320 entries with tagged alchemy_category but low confidence.

Similar to the isNews labelling, the ? values can fall into any one of the given categories which can be inferred pretty quickly given the context. This again leads us to the point of either manually filling it in or creating a separate NLP classifier.

Numerical Columns

None of the columns seem to have any invalid values and these numeric columns were given to the StandardScaler for standardization before model training.

Correlation analysis

Following were the highly correlated columns,

- AvglinkWithOneCommonWord, AvglinkWithTwoCommonWord => 0.81
- AvglinkWithOneCommonWord, AvglinkWithThreeCommonWord => 0.56
- AvglinkWithTwoCommonWord, AvglinkWithThreeCommonWord => 0.76
- AvglinkWithThreeCommonWord, AvglinkWithFourCommonWord => 0.85
- AvglinkWithTwoCommonWord, AvglinkWithFourCommonWord => 0.56
- embedRatio, redundancyMeasure => -0.89

Of these the columns that were dropped are as follows,
AvglinkWithTwoCommonWord
AvglinkWithThreeCommonWord
embedRatio (it was dropped because its meaning is ambiguous as compared to redundancyMeasure)

websiteName Feature Engineering

Using the URL parameter to extract the domain name/website name out of it and create a new categorical column called “websiteName”.
The following images will describe the entire analysis,

```
website_names = non_correlated_data['websiteName'].value_counts()

print(len(website_names[website_names > 0]))
print(len(website_names[website_names > 5]))
print(len(website_names[website_names > 10]))
print(len(website_names[website_names > 15]))
print(len(website_names[website_names > 20]))
print(len(website_names[website_names > 30]))
```

```
3372
199
68
40
31
19
```

There are in total 3372 unique website names in around 7000 entries.

We have 19 unique website names that have count > 30 in the dataset,
so we can let these website names be as it is and combine all the other website names into "other" category

So in total there will be 20 categories in total in websiteName

Following are the 19 unique categories,

```
websitesWithAtleast30Entries = list(website_names[website_names > 30].index)

website_names[website_names > 30]
```

```
www.insidershealth.com       143
sportsillustrated.cnn.com    109
www.huffingtonpost.com        99
allrecipes.com                93
bleacherreport.com            86
www.youtube.com               85
blogs.babble.com              62
www.ivillage.com              59
www.foodnetwork.com           57
www.dailymail.co.uk           46
www.epicurious.com            36
www.womansday.com             35
www.bbc.co.uk                 34
www.guardian.co.uk            33
www.popsci.com                33
www.marthastewart.com         33
www.collegehumor.com          31
www.buzzfeed.com              31
itechfuture.com               31
```

NLP Part 1: TF-IDF Vectorizer

Webpage Description had already been processed as described in the previous section.

The following NLP preprocessing steps were done on the text data,

1. Remove HTML
2. Remove non-letters
3. Convert to lower case, split into individual words
4. Remove stop words
5. Stem or Lemmatize the words (option is provided to choose which one)
6. Join the words back into one string separated by space so that it can be sent to a vectorizer.

This preprocessed data was directly sent to a TF-IDF vectorizer and the output vocabulary contained around 58000 words and combining it with our existing dataset got us to a total of around 58100 columns.

In order to reduce the number of columns, i.e. reduce the vocabulary size, we used the “max_features” parameter of the TF-IDF Vectorizer.

If max_features = x, build a vocabulary that only considers the top x words ordered by term frequency across the corpus.

Feature Selection: Chi-Squared Test

Previously we had around 80,000 words in the vocabulary for vectorization which we reduced to 10,000 using the max_features parameter of the vectorizer for reducing redundancy as well as increasing the score.

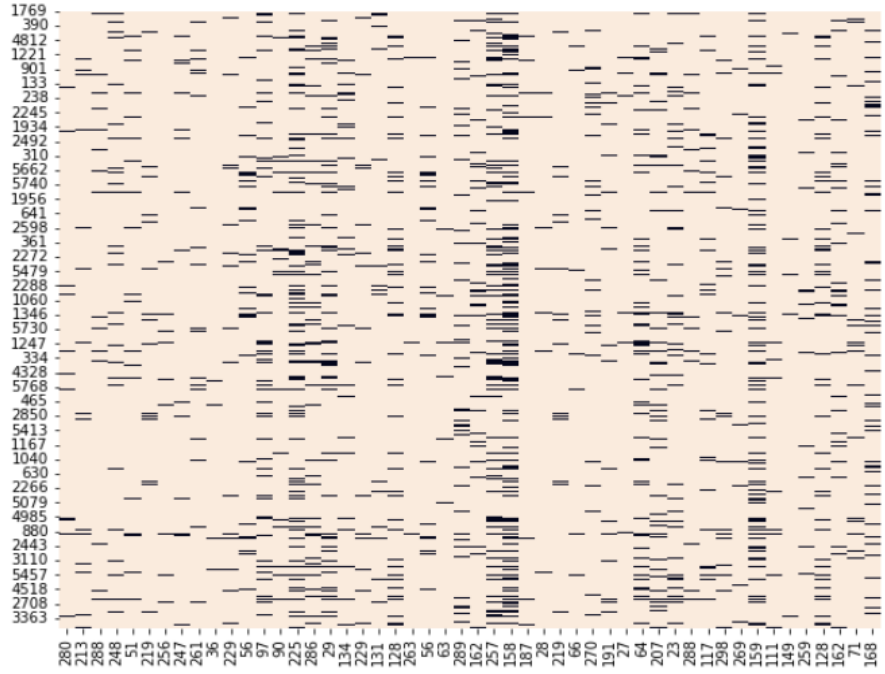
The Chi-Squared Test can be used to reduce this even further.

The test takes two parameters, an input feature and a target variable and outputs a score (p-value) which represents the confidence that the target variable is independent of the input feature (NULL Hypothesis).

A p-value of 0.05 is typically considered statistically significant enough for the target variable to be dependent on the input feature.

So we kept the p-score threshold at 0.05 and ran the Chi-Squared test considering each word in the vocabulary against the target variable.

Only 305 words out of the 40,000 words vocabulary being considered had p-value <= 0.05
So this was our new vocabulary that we went ahead with.
Heatmap of new vectorized data



Can see just how dense this vectorized data is as compared to the usual sparse TF-IDF vectorized matrix.

The new vectorized data was then used to train a Logistic Regression model with base parameters and this actually gave us a score of 0.86302 which is the same as our previous Logistic Regression model with 10,000 words in the vocabulary.

So even though the accuracy didn't improve, there was a significant improvement in terms of memory used, time required for training and scalability as well.

Experiments

Upto Evaluation 1

1. Dropped the "?" value columns and trained the default value Logistic Regression model on this data => Kaggle Score approximately 0.66

2. Webpage Description column was used with TF-IDF vectorizer and a max vocabulary size of 10,000 words to train a base Logistic Regression model => Kaggle Score approximately 0.81

3. The "?" values were filled in with values which are described below,
isNews & isFrontPage News => All '?' values were given new category of 'unknown'

alchemy_category & alchemy_category_score => All ‘?’ values were assigned to ‘unknown’ category and corresponding alchemy_category_score was assigned to 1.

4. The filled in data of experiment 3 was combined with the vectorized data of experiment 2 to train a Logistic Regression model => Kaggle Score approximately 0.79

5. Different values were tried for the vocabulary size of TF-IDF vectorizer and trained on the Logistic Regression model but no improvement in score was observed.

6. Stemming & Lemmatization both were alternatively used in each training method for processing webpageDescription but there was significant improvement or deterioration of the score.

Upto Evaluation 2

1. Chi-Squared Test

2. Manual Feature Selection
   - Manually chose subsets of features from the dataset in addition to the 305 features obtained after the Chi-Squared test.
   - These subsets were trained on the base Logistic Regression model and the test ROC scores were recorded.
   - Tried adding the URL column to our best score submission.

The following set of features gave the best score,
webpageDescription; avgLinkWordLength; AvglinkWithOneCommonWord; AvglinkWithFourCommonWord; redundancyMeasure; frameTagRatio; isNews; lengthyDomain; hyperlinkToAllWordsRatio; alphanumCharCount; linksCount; wordCount; spellingErrorsRatio

However this didn’t translate to a better Kaggle score as these features would more often than not reduce the score when tried with any of the typical sklearn models.

Upto Evaluation 3

Hyperparameter tuning with TF-IDF vectorized data with 10,000 words in the vocabulary
Conclusion: Not much of an improvement over just using default Logistic Regression

Word2Vec

Approach 1:

Using each unique webpageDescription input as a sentence in the “sentences” parameter of the Word2Vec model.

The complete parameter list is as follows,

```
vector_size = 300       => Word vector dimensionality
min_count = 1           => Minimum word count
workers = 4             => Number of threads to run in parallel
window = 10             => Context window size
sampling = 1e-3         => Downsample setting for frequent words
sg = 1                  => Use Skip-Grams instead of Continuous Bag of
Words
```

Approach 2:

Each tokenized webpageDescription is marked for trigrams data with the help of the Phrases structure provided by gensim.

Sentences => Phrases => Bigrams output => Phrases => Trigrams output

This trigrams data is then sent to the Word2Vec moTraining the Word2Vec model gives us a word vector for each word in the vocabulary.

To convert a webpageDescription into a feature vector we use the following Logic,

```
sum = np.zeros(300,1); # 300 = size of word vector

for each word in webpageDescription,
      - Check if we have a word vector for it
      - If we do then add it to "sum"
      - Keep track of count of such words

sum = np.divide(sum, "no. of words for which we had word vector");
```

So each webpageDescription's feature vector is the average of the word vectors of the individual words in the description for which we have a word vector in the Word2Vec model.

Both Word2Vec approaches were trained on the base Logistic Regression model.

Approach 1 gave Kaggle score of 0.8796
Approach 2 which is the Trigrams approach gave Kaggle score of 0.8834
Approach 2 seemed to be the more superior one so it is what we went ahead with.

Final Experiment: Training Word2Vec Approach 2 on MLPClassifier with Hyperparameter Tuning to find the best architecture.

Models Used

1. Logistic Regression
2. Random Forest
3. SVM
4. KNN
5. Multinomial Naive Bayes (not in Word2Vec approach because of negative values)
6. MLP Classifier

Models and their Scores

	Model	**Kaggle Score**
TF-IDF Vectorizer with 10,000 words and Hyperparameter tuning on sklearn model	Logistic Regression	0.8642
	Multinomial Naive Bayes	0.8560
	Random Forest	0.8587
	SVM	0.8621
	KNN	0.5502
Word2Vec with Trigrams Approach + Word Vector Averaging + Hyperparameter tuning on sklearn model	Logistic Regression	0.8834
	Random Forest	0.8826
	SVM	0.8662
	MLPClassifier	**0.8870**

Hyperparameters for best performing MLPClassifier,

```
{
      'activation': 'relu',
      'alpha': 0.05,
      'hidden_layer_sizes': (100, 50, 100, 50),
      'learning_rate': 'adaptive',
      'solver': 'sgd'
}
```

Architectures tried in hyperparameter tuning were,

```
'hidden_layer_sizes': [
      (50,50,50), (50,100,50), (100,), (100, 50, 50), (100, 50, 50, 50)
      (100, 50, 100, 50), (100, 50, 100, 50, 50),
      (100, 50, 100, 50, 50, 50),
      (100, 50, 100, 50, 50, 100, 50, 50),
      (100, 100, 100, 100, 100, 100, 100, 100, 100, 50),
      (100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 50)
]
```

Individual Contributions

Jasvin James - MT2021059

Chi-Squared Testing
Manual Feature Selection
Word2Vec modelling
Using MLP Classifiers

Khushal Abrol - MT2021063

Preprocessing
Prototype Modelling
Hyperparameter Tuning on all models

Conclusions

When you have a text column in your dataset, it is best to focus your full efforts on that single column instead of trying to make something out of the other ones :P

Word2Vec clearly performed the best in this use-case, although it may not always be better than even TF-IDF vectorizer. It all depends on the problem statement.

MLPClassifier tuning was done upto 10 hidden layers but even then only the 4 hidden layer architecture was chosen. So more logic must go behind into NN architecture designing than just randomly throwing numbers at the tuning process.

There are many more approaches to convert a collection of word vectors into a feature vector for a dataset, the averaging approach is just one of them.

The extra context that trigrams marked data provided seems to have helped with the prediction as it resulted in overall increase of score.

SVM performed surprisingly poorly at times and KNN gave such horrible performance that no more attempts were made to salvage it.

References

[1] - NLP Processing Reference:
https://www.kaggle.com/c/word2vec-nlp-tutorial/overview/part-1-for-beginners

[2] - Word2Vec Introduction
https://www.kaggle.com/c/word2vec-nlp-tutorial/overview/part-2-word-vectors

[3] & [4] - Understanding Word2Vec
https://medium.com/swlh/sentiment-classification-using-word-embeddings-word2vec-aedf28fbb8ca
https://jalammar.github.io/illustrated-word2vec/

[5] - Trigrams approach to Word2Vec
https://www.kaggle.com/alexcherniuk/imdb-review-word2vec-bilstm-99-acc