FeatureRank

Kuhan Wang[1]

1. Insight Data Science

October 2, 2015

Abstract

FeatureRank is a software tool for extracting correlations between text ngram features and user engagement, thereby optimizing the placement of financial widgets on URL articles.

1 Directory Structure

- /

  `processing.py`

  Pre-processing to parse relevant information from engagement csv files.

  `crawl.py`

  A simple web crawler that pulls the title and $< p >$ tag text from URLs.

  `FeatureRank.py`

  Driver file to execute main functions.

  `feature_extraction_model.py`

  The core program that contains the machine learning algorithms.

  `post_processing.py`

  Post processing to produce evaluation metrics and ngram rankings.

  `web_text_data_set_1_2.json`

  A file containing the sorted JSON dictionaries of each URL, this is the input to FeatureRank.

  `read_json.py`

  A script converting the JSON file into a format that can be read into the model learning functions.

- /DATA - engagement data is stored here.
- /RESULTS - output data is stored here.
- /DOC - the documentation is stored here.

The pipeline is designed so that the index and web scraping functions are separate from the machine learning. This way the user can could utilize his or her own web crawling tool and convert the results into JSON format to use with `FeatureRank.py`. The pipeline execution is summarized below .

1. Summarize the engagement data from SCORE.CSV, ENGAGEMENT.CSV and CAPTIVATE.CSV by using `processing.py` which produces an indexed csv file of all the URLs.
2. Input the index csv file to `crawl.py` and crawl each website. The output is a series of JSON dictionaries contained in `web_text_xxxx.json` that contains the textual information of the URL and the corresponding engagement data.
3. `web_text_xxxx.json` is the input data to `FeatureRank.py` which drives the supervised learning algorithm and will output the relative ngram importance using a bag of words approach.

1.1 FeatureRank

The input data is held in a series of JSON dictionaries. The file `web_text_xxxx.json` contains a series of dictionaries (one for each URL). The format per URL is as follows:

```
{
“body”: [ “blah blah blah1.”,
“blah blah blah2.”,
“blah blah blah3.”,
...],
“url”: “http://www.engineercents.com/”,
“type”: “careercalculator\r \n”,
“pageloaded”: 169.0,
“title”: “Engineer Cents - Financial Independence By 35 Or Bust!”,
“widgetviewed”: 104.0,
“widgetused”: 16.0,
“onerank”: 2.333333602329252
}
```

The body of the text is stored as strings in a PYTHON list separated by commas. With a few exceptions the remaining terms are self-explanatory. pageloaded, widgetviewed, and widgetused represent the website traffic and engagement performance of the highest scored widget as obtained from ENGAGEMENT.CSV and onerank indicates the score that widget received.

1.2 Parameters

The tool takes the following parameters:

- - -h: Display usage information .
- -json: The input JSON data file.
- -w: The widget type to analyze, names are taken from CAPTIVATE_KEYWORDS.CSV.
- -nL : The lower bound of ngram ranges to examine, minimum value of 1.
- -hL: The upper bound of ngram ranges to examine, increasing this value beyond two can dramatically slow the code.
- -mDF: The minimum cutoff for ngrams with respect to frequency in the entire corpus of documents. Higher values includes less features, lower values more. Values between are reason $0.02 - 0.05$. Setting values < 0.02 will result in an exponential increase in the feature space and thereby slow computation.
- -pLoad: Exclude URLs that loaded less than pLoad times.
- -wView: Exclude URLs where the widget was viewed less than wViewed times.
- -Seed: Set the random seed, useful to test random sampling methods.
- -f: Set the percentage of the data to retain for testing purpose. Set to 0 in order to learn on all data.

2 Example Usage

Once the necessary packages are installed the model can be executed like:

```
python FeatureRank.py -json web_text_data_set_1_2.json \
-w budgetcalculator -nL 1 -nH 2 -mDF 0.02 -pLoad 5 -wView 1 -Seed 1 -f 0
```

The script `feature_extraction_model.py` will take the body of the JSON dictionary as the input training matrix X, the engagement rate is calculated as #Widget Clicked/#Widget Viewed and binarized such that non-zero values are 1, this is the target vector, y.

The X matrix is utilized to build a feature matrix consisting of word counts using the CountVectorizer class. The columns of the feature matrix are the ngrams of the entire corpus while the rows are the number of URLs (data points).

The feature matrix is then normalized by inverse document frequency weighting using the TfidfTransformer class. The output matrix X_{IDF} is the trained against the binarized engagement target y using a supervised learning model. Three methods are currently implemented:

- Random Forest Classifier,
- Logistic Regression,
- Stochastic Gradient Descent with a hinge loss function.

The recommended model which has been studied for the consulting project is Logistic Regression[1]. The user can modify the parameters of each model by accessing `feature_extraction_model.py`.

Regardless, the relevant output of any model are the coefficients of the model. These represent the relative importance of the ngrams (columns of X_{IDF}) as every ngram is associated to a model coefficient.

The parameters of the model and the predicted model results are passed to `post_processing.py` which output a number of metric scores[2]. The output files,

- VALIDATION_*nL_nH_mDF_pLoad_wView_Seed_f*.CSV,
- RANKED_WORDS_NGRAM_*nL_nH_mDF_pLoad_wView_Seed_f*.TXT,

will be found in RESULTS/W

When the code is ran with f!=0 a number of predictive scoring metrics such as the precision and confusion matrix will be written to the validation file. The file RANKED_WORDS_NGRAMS_XXXX.TXT contains the ngram features of the corpus ranked by their associated coefficient terms and by model. Negative values indicate anti-correlation to engagement.

In addition, `read_json.py` will output a file containing the body of each JSON dictionary that has been matched to the corresponding widget in the root folder, BODY_TEXT_W.TXT.

3 Environment and Packages

The code is built using Python 2.7.6 on a Linux Mint 17.2 distro machine. The compiler is gcc 4.8.2. The PYTHON libraries needed to execute `FeatureRank.py` are,

- SCIPY,
- NUMPY,
- JSON,

In addition, the index and web crawling scripts use,

- LIBURL2,
- BEAUTIFULSOUP.

[1]Stochastic Gradient Descent also demonstrates reasonable performance. Random Forest is relatively poor in comparison.

[2]These are the precision, recall, accuracy and full confusion matrix of the prediction.

4 Known Issues and Nuances

- Some URLs are present in SCORE.CSV but not present in ENGAGEMENT.CSV or vice versa, these are not included as data points.
- The list of unigram stop words must be curated very carefully, words may be selectively removed from the corpus by reiterating the pipeline multiple times but this should be checked against the corpus.
- To remove "stop words" that are bigrams or longer strings, they must be manually removed by adding them to the appropriate list in `read_json.py`.
- A number of URLs forbid web crawling using LIBURL2, this is roughly ~ 5 of the dataset.
- The model is only applied to the budgetcalculator, assetallocationcalculator, homeaffordabilitycalculator and careercalculator. The remaining widgets lack a quantity of data. The user can still use the pipeline but the results are not guaranteed to be reliable.
- Data should be added to the model such that the classes are balanced. This means repeatedly adding URL data without engagement will not greatly improve the model learning.

5 Assumptions

- When combining different data sets, they do not overlap. I.E. if the engagement data from two separate periods are combined the same URL is not repeated[3].
- The mobile and desktop versions of the same URL are treated as separate data points, the engagement is not combined together, even if the same widget was placed on both versions of the URL.
- Because the overall engagement is low, the engagement rates are binarized. If the engagement rate is not 0, it is 1. Therefore, the engagement target represent simply: "did the user click or not click on the widget".
- The relatively low engagement means the data is imbalanced. For a particular widget, if the engagement data is imbalanced, the data is downsampled until the classes (0 and 1's) are balanced to $\sim 50\%$. If the widget data is imbalanced towards engagement (more 1's then 0's) no action is taken. This only occurs for the homeaffordabilitycalculator.
- Unigram features with the same stem but different suffices are currently treated differently (for instance "word" and "words"). A future version could change this by lemmatizing each unigram feature of every URL. This will be more computationally demanding.

[3]This only concerns the two datasets I received in September, which I combined and used together.