NPL Text classification model on US clickbait and non-clickbait requirements

GOALS

• To perform a text classification experiment and examine the results

• To become familiar the scikit-learn machine learning library

• Optional exercise: To gain hands-on experience with word embeddings

DATA

For this project you will build a text classification model to identify clickbait, or text headlines whose main purpose is to attract attention and entice readers to follow a link.

You will use data that has been created and shared by other NLP researchers on GitHub:

• Positive examples: https://github.com/pfrcks/clickbait-detection/blob/master/clickbait

• Negative examples: https://github.com/pfrcks/clickbait-detection/blob/master/not-clickbait.

Phase 1 – Reading the data

• Using Python, read in the 2 clickbait datasets (See section DATA), and combine both into a single, shuffled dataset. (One function to shuffle data is numpy.random.shuffle)

• Next, split your dataset into train, test, and validation datasets. Use a split of 72% train, 8% validation, and 20% test. (Which is equivalent to a 20% test set, and the remainder split 90%/10% for train and validation. If you prefer, you may save each split as an index (list of row numbers) rather than creating 3 separate datasets.

• What is the "target rate" of each of these three datasets? That is, what % of the test dataset is labeled as clickbait.

Phase 2 – Training a single Bag-of-Words (BOW) Text Classifier

Using scikit-learn pipelines module, create a Pipeline to train a BOW naïve bayes model. We suggest the classes CountVectorizer and MultinomialNB. Include both unigrams and bigrams in your model in your vectorizer vocabulary (see parameter: ngram_range)

• Fit your classifier on your training set

• Compute the precision, recall, and F1-score on both your training and validation datasets using functions in sklearn.metrics.

Phase 3 - Hyperparameter Tuning

Using the ParameterGrid class, run a small grid search where you vary at least 3 parameters of your model

• max_df for your count vectorizer (threshold to filter document frequency)

• alpha or smoothing of your NaïveBayes model

• One other parameter of your choice. This can be non-numeric; for example, you can consider a model with and without bigrams (see parameter "ngram" in class CountVectorizer)

Show metrics on your validation set for precision, recall, and F1-score.

Phase 4 - Model selection

Using these validation-set metrics from the previous problem, choose one model as your selected model. It is up to you how to choose this model; one approach is to choose the model that shows the highest F1-score on your training set.

Next apply your selected model to your test set and compute precision, recall and F1

Phase 5 - – Key Indicators

Using the log-probabilities of the model you selected in the previous problem, select 5 words that are strong Clickbait indicators. That is, if you needed to filter headlines based on a single word, without a machine learning model, then these words would be good options.

Phase 6 - Regular expressions

IT department has reached out to you because they heard you can help them find clickbait. They are interested in your machine learning model, but they need a solution today.

• Write a regular expression that checks if any of the keywords from the previous problem are found in the text. You should write one regular expression that detects any of your top 5 keywords. Your regular expression should be aware of word boundaries in some way. That is, the keyword "win" should not be detected in the text "Gas prices up in winter months".

• Using the python re library – apply your function to your test set. Show
is the precision and recall of this classifier?

Phase 7 - Comparing results

Compare your rules-based classifier from the previous problem with your machine-learning solution. Which classifier showed the best model metrics? Why do you think it performed the best?

If you had more time to try to improve this clickbait detection solution, what would you explore?