Report.md

Bias Detection in News

Bias Detection in news articles. Detect sentence level and article level bias in news domain.

The task performed is bias detection by-article.

The reference was taken from the paper: https://aclanthology.org/S19-2163.pdf

Dataset

SemEval 2019 dataset on news bias: https://zenodo.org/record/1489920

The data is split into multiple files. The articles are contained in the files with names starting with "articles-" (which validate against the XML schema article.xsd). The ground-truth information is contained in the files with names starting with "ground-truth-" (which validate against the XML schema ground-truth.xsd).

The data (filename contains "byarticle") is labeled through crowdsourcing on an article basis. The data contains only articles for which a consensus among the crowdsourcing workers existed. It contains a total of 645 articles. Of these, 238 (37%) are hyperpartisan and 407 (63%) are not.

Data cleaning

- The XML file `articles-training-byarticle-20181122.xml` is loaded `data` and the XML file `ground-truth-training-byarticle-20181122.xml` is loaded into `labels`.
- This data is then split into train and test data.
- nltk.tokenize library was used to tokenize into sentences and then into words.
- RegexpTokenizer is also used from this library.
- All the words are converted into lower case.
- For CBOW:
  - After we get the document vector (explained in `Core concept` sub-section), we need to fill NaN values for which we use forward fill followed by backward fill (this is to fill initial NaN values). Then the remaining null values are replaced with 0.
  - We also dropped the columns which didn't pass the threshold criteria for minimum number of non-null values (which in our case was 1/4 of no. of rows).
- For LSTM and elmo
  - First of all the values with type none are excluded which reading the input.
  - Now for each sentences only that word is taken which does not have any ascii charachter other than alphabets.
  - All the caplital letters are converted to smaller case.

Core concept

- First we are doing data cleaning. As data contains lot of ascii characters like(?,*,()= etc which need to be removed.
- Then we calculated word embeddings using three models(each in one file) => CBOW , LSTM , elmo
- After that we fine - tuned that model on our dataset.
- Now that we have embedding of each word in our dataset we need to find feature vector and corresponding labels for supervised learning. So feature vector for an article is calculated as follows -
  - First of all for each sentence a matrix is calulated whose each column is word embedding of the words in that sentence
  - After that average is taken over whole matrix to calculate a single value for a sentence.
  - Now for a document a list is formed where each element of the list is the value of the sentence .
  - So we have a list for each article which is the feature vector of that particular article and we have the label for it - 0 for unbiased and 1 for biased article.
- After that training is done over the training dataset using any classification algorithm.We have used two classifications algorithm one is random forest and other one is SVM.

- Finally the trained model is tested on testing dataset and the accuracy is calculated.

Baseline

---

1) Vanilla CBOW

The embeddings were not loaded from anywhere but randomly initialized and purely run on our CBOW code which is written from scratch.

Parameters

- CBOW parameters

```
V = len(words_without_repitions)
num_iters = 50000
N = 100
alpha = 0.03
Test_data : Train_data = 1:9
Train data = 580
Test data = 65
```

- Finally for the article we chose 32 columns (Keep only the columns with at least 1/4 non-NA values) and we had 580 rows (train_data).
- Parameter tuning such as train-test split ratio, filling NaN values with diff methods, different classifiers, different padding values was done to maximize the overall scores.

Scores

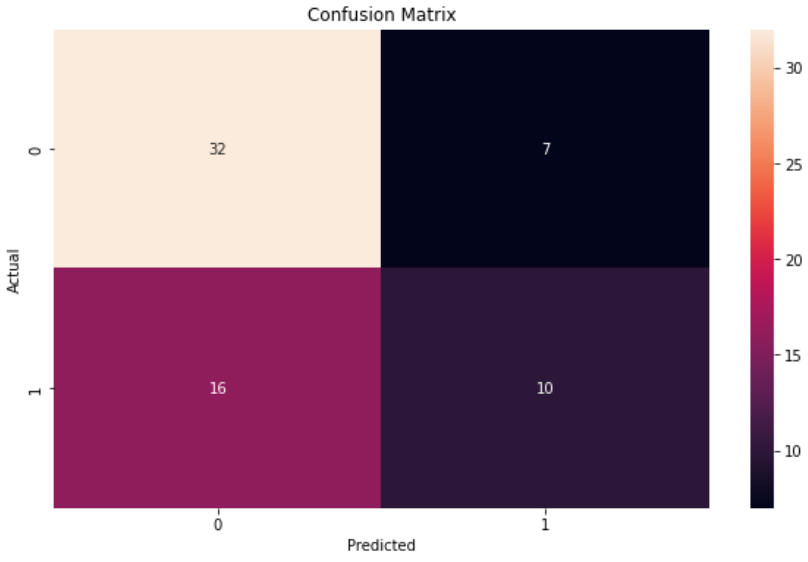
- Accuracy: 64.61538461538461 %
- F1 Score: 46.51162790697674 %
- Classification Report:

```
              precision    recall  f1-score   support

           0       0.67      0.82      0.74        39
           1       0.59      0.38      0.47        26

    accuracy                           0.65        65
   macro avg       0.63      0.60      0.60        65
weighted avg       0.64      0.65      0.63        65
```

- Confusion Matrix:



The classifier used was **RandomForest** which gave the best results. The hyper-parameter tuning was done with the help of **RandomizedSearchCV**.

The best hyper-parameter set was found to be:

```
RandomForestClassifier(max_depth=12, max_features='sqrt', n_estimators=900,
                       random_state=48)
```

---

i) Gensim embeddings

Our CBOW model outperformed the gensim embeddings (*GoogleNews-vectors-negative300.bin*) score to find out bias. The gensim model gave the following scores:

- Accuracy: 55.38461538461539 %
- F1 Score: 29.268292682926834 %
- Classification report:

```
              precision    recall  f1-score   support

           0       0.60      0.77      0.67        39
           1       0.40      0.23      0.29        26

    accuracy                           0.55        65
   macro avg       0.50      0.50      0.48        65
weighted avg       0.52      0.55      0.52        65
```

---

ii) Glove embeddings

Our CBOW also outperformed the **GloVe embeddings** (http://nlp.stanford.edu/data/glove.6B.zip) scores. The scores are:

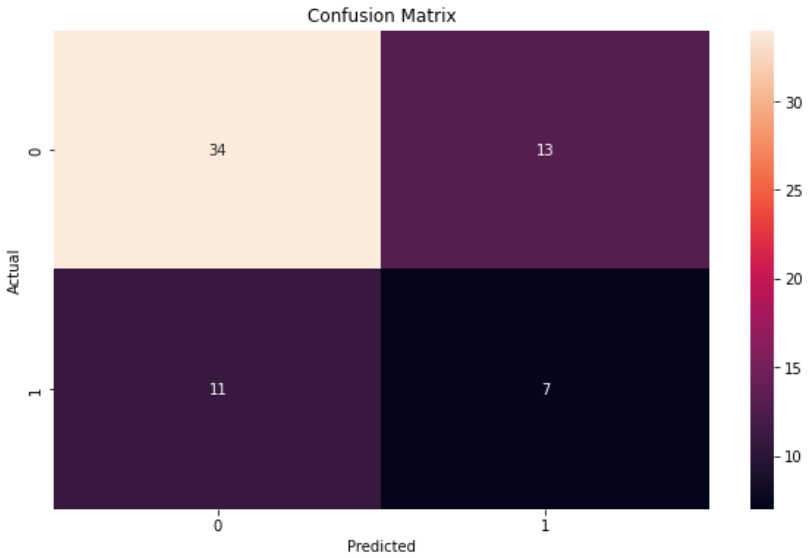
- Accuracy: 63.07692307692307 %
- F1 Score: 36.84210526315789 %
- Classification report:

```
              precision    recall  f1-score   support

           0       0.76      0.72      0.74        47
           1       0.35      0.39      0.37        18

    accuracy                           0.63        65
   macro avg       0.55      0.56      0.55        65
weighted avg       0.64      0.63      0.64        65
```

- Confusion matrix:



2) LSTM

Second approach that we used is LSTM . Since CBOW doesn't use long term context, therefore LSTM (Long short term memory) is used which is designed in such a way that solves vanishing gradient problem and handles long context dependencies.

In this we have used pre-trained word2vec and then fine tuned on our dataset using LSTM layer . After that we have used random forest classification and SVM to check for the scores on our trained model.

- Hyper-Parameters

```
num_iters = 100
Embedding_size = 100

Test_data : Train_data = 2:8
Train data = 400
Test data = 100
```

Scores

- Accuracy: 0.73 %
- Classification report:

```
              precision    recall  f1-score   support

           0       0.73      0.82      0.84       102
           1       0.27      0.19      0.15        38

    accuracy                           0.73       140
   macro avg       0.36      0.50      0.42       140
weighted avg       0.53      0.73      0.61       140
```

Baseline+

ELMO

So in this we have used elmo embeddings. Elmo is advaned technique to calculate embeddings as it uses Bi-LSTM to get forward as well as backward context. Character-level tokens are taken as the inputs to a bi-directional LSTM which produces word-level embeddings.Therefore it handles out of context words as well.Therefore elmo is used as SOTA approach in calculating word embeddings.

- Hyper-parameters used -

```
Embedding_size = 1024
maximum_length_sequence = 140
Test_data : Train_data = 2:8
Train data = 200
Test data = 50
```

For classification part we have used random forest with parameters -

```
num_trees = 1000
n_features = 100
n_depth = 5
```

- Accuracy: 75.51020408163265 %