Sarcasm and Irony Detection

Team Members: Aditya Raghuvanshi, Satya Swaroop Gudipudi, Rohan Chowdary  
Team Number: 15  
  
Description:  
As part of the project, we have explored various approaches and in this report we walkthrough regarding datasets we used, our approaches, our findings and next steps. We will also recap our committed timelines as reported in project outline.

Approach:

We have identified benchmark datasets for which baselines are available and performed exploratory data anlaysis to identify any patterns and applied tokenization by ensuring we filter noise in the data and finally we applied embeddings to convert text to meaningful representation. The predicted scores from the trained model are evaluated with metrics like F1, AUC, Accuracy against baseline results.

# Exploratory Data Analysis(EDA) and Model results:

We identified two datasets: SemEval -2018 for irony detection and SARC (Reddit politics) dataset for sarcasm detection.

### SemEval-2018 exploration:

The dataset is gathered from twitter and hence it has hashtags, urls, slang and emoticons with various expressions. The dataset size is relatively small and hence challenging to train complex model architectures like transformer model from scratch.

Train Data: 3817

Test Data: 784

Vocab: 10803

Max length of tweet observed in 0.95 quartile: 231

No. of hashtags in vocab: 2769

Preprocessing steps applied:

Since it is twitter data, an exhaustive preprocessing and cleanup steps are implemented like

* Replace numbers with num
* Replace URLs with URL
* Replace unknown words as UNK
* Replace unknown hashtags as #hashtag
* Replace ranges patterns like 2-12 etc with range
* Replace alphanumerics like 2ve etc as alphanumeric
* Replace amounts patterns like 136k, 500k etc as amount
* Replace versions patterns like 4.0, etc as version
* Replace temperature patterns like 100.20, 971.2 etc as temperature
* Replace time patterns like 3:30, 1am, etc as time
* Replace years like 2003 etc as year
* Replace selective punctuation as punctuation is especially important for irony detection.
* Replace emoticons which are not available in emoji\_vec with single word descriptions.
* Replace symbols with symbol
* Replace some special characters as special
* Inserted additional token ‘[EMOTICON]’ for tweets having emotions.
* Inserted additional token ‘[ELONGATED]’ for tweets having elongated words like “Aweeesome”, “Foreverrr”, etc.

Also created a hashtag categorization for common hashtags like abusive common words replaced hashag with #abusive and similarly with #sarcasm, #racial, etc and this is done only for word2vec pre-trained only as fasttext was able to handle unknown vocab.

#### Model Training approaches for irony detection:

Since the dataset has untraditional language with lot of expressions, slang, spell errors, emoticons etc noise a pre-trained static embeddings would not have equivalent representations and a traditional tokenizer would fail to tokenize.

Tokenizers: We played with various tokenizers like TweetTokenizer from nltk and also trained subword (BPE) tokenizer. These tokenizers are robust in handling such noisy data.

Pre-trained embeddings: We applied Many to one Sequence to Sequence modelling and used text representations from word2vec\_twitter\_model and fasttext\_twitter\_raw. Fasttext is more robust as it is character based and able to provide representation even for unknown words. For emoticons we applied embeddings from emoji2vec.bin.

The challenge is word2vec and fasttext gives 400 dimensions vectors as word representation and emoji2vec provides 300 dimensions vector representation for given emoticon. This had to be handled in model training by filling the last 100 dimensions with null values/random values.

As the dataset size is small applied k-cross validation and following are results from experiments and also listed the baseline results which we will be comparing throughout this project:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| #Exp No | Model | Description | Accuracy | F1 | AUC |
| 1 | BiLSTM with attention with word2vec embeddings | hid\_dim: 256, num\_layers:2, lr: 0.001, Adam optimizer | 0.4337 | 0.5723 | 0.5858 |
| 2 | BiLSTM with attention with fasttext embeddings | hid\_dim: 256, num\_layers:2, lr: 0.001, Adam optimizer | 0.5255 | 0.6092 | 0.6638 |
| 3 | Transformer Encoder | emsize = 200  d\_hid = 200  nlayers = 2  nhead = 2  dropout = 0.1 | 0.6454 | 0.6128 | 0.7028 |
| 4 | Bidirectional encoder | hidden\_size=256, num\_attention\_heads=4,  num\_hidden\_layers=2, intermediate\_size=1024, | 0.6429 | 0.5018 | 0.6619 |
| 5 | Bert BiLSTM | Bert\_dropout = 0.1, LSTM\_units = 512, dense\_units = 50 | 0.61 | 0.16 | 0.673 |
| 5 | Bidirectional Encoder with Exponential Positional Encoding | d\_model = 768, Bert configurations | 0.719 | 0.69 | 0.81 |
| 6 | SetFit with sentence-t5-base body and voting classifier head | Num\_epochs = 1, num\_iterations = 25 | 0.769 | 0.746 | 0.85 |
| 7 | Baseline (RCNN – Roberta) | Baseline | 0.82 | 0.80 | 0.89 |

Notes:

* The hyperparameter tuning is performed with optuna for 50 trials and the trial results can be found in the code submitted along.
* Also the code has K-cross validation results for each fold.
* The Bidrectional encoder refers to BERT architecture only but we are not using bert model weights nor finetuning on top of bert model.
* We can clearly see the pretrained embeddings has good effect on the model training fasttext compared with word2vec, this means if we replace pretrained embeddings with deep contextualized embeddings like ELMo the results could be even more promising.

### Sarcasm Dataset:

Since the Irony dataset is very small, the complex models like transformer based are often overfitting. For experimentation we took Sarcasm dataset (Reddit politics) data for which benchmarks are available.

Train Dataset: 31596

Validation Dataset: 7900

Max length observed in train data: 504

#### Model architecture approach and results:

The dataset is having reddit comment and parent comment. The parent comment could serve as context. We concatenated both the comment and parent\_comment with “[SEP]” token as delimiter for transformer model to distinguish and consider as sentence pair training. Following are the results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| #Exp No | Model | Description | Accuracy | F1 | AUC |
| 1 | Bidirectional encoder | hidden\_size=512,  num\_attention\_heads=4,  num\_hidden\_layers=3,  intermediate\_size=2048,  hidden\_dropout\_prob=0.1,  attention\_probs\_dropout\_prob=0.1, | 0.6866 | 0.7644 | 0.7277 |
| 2 | Baseline (RCNN – Roberta) | Baseline | 0.79 | 0.78 | 0.85 |

\* The test data is not available hence had to take stratified data for comparison results.

# Various Model architectures explored:

We have explored following various model architectures for our experiments and applied some novel techniques to achieve reasonable results.

1. Finetuning/Transfer-Learning
2. Few Shot Sentence Transformers training with SetFit
3. Ensemble modelling
4. A Novel task specific positional encoding techniques like inverse exponential decay.
5. Combination of transformer based model and sequence based models such that transformers for learning better representation and sequence models to learn the sequential pattern in the data.

Summary:

We started with basic implementations like BiLSTMs as baseline and incrementally experimented with simple to complex architectures like task specific positional encoding, fine-tuning, transfer learning, few shot training and fusion models, where we improved the auc from 0.585 to 0.85. We are able to achieve almost equivalent results of benchmark[1] and our novel model architecture implementations are beating some of the baselines mentioned in [1]. We have applied experiments on Irony and Sarcasm datasets and compared with benchmarks. We also successfully trained irony and sarcasm detection for code mix “Hinglish” data.  
We have studied the data and existing available implementations and able to identify valid observations from published papers[1].

# Code artifacts:

The model artifacts are available here:  
<https://drive.google.com/drive/folders/1wwpnXvfuH1vbCFFsTfMj_xuE1fhlSRZd?usp=drive_link>  
And the code is attached.

References:

1. [SemEval-2018 Results](https://link.springer.com/article/10.1007/s00521-020-05102-3/tables/2)
2. [Reddit Politics Dataset results](https://link.springer.com/article/10.1007/s00521-020-05102-3/tables/3)