Introduction to NLP

Domain Terms Extraction: Project Report

Team Name: NLoPq **Team Number: 34**

Archit Jain (2019101053) Pulkit Gupta (2019101078)

Problem Statement

We need to scrape multiple articles from multiple domains for creating a corpus and then using this corpus, we need to build a dictionary of domain-specific terms including multiword expression, based on how frequent they occur with each other.

Literature Review

- **An Unsupervised Approach to Domain-Specific Term Extraction** [1]

  This paper uses an unsupervised method to extract domain-specific terms from the Reuters document collection using TF-IDF (term frequency and inverse document frequency) over domains. All the terms are scored using TF-IDF and those terms with scores higher than the threshold which is selected based on score distribution (choosing the point at which there is a maximum drop in TF-IDF scores). They had manually verified how well their method extracted domain-specific terms.

- **Unsupervised Technical Domain Terms Extraction using Term Extractor** [2]

  In this paper, they have used the pre-built dataset “Term Traction ICON-2020” consisting of four domains namely Biochemistry, Communication, Computer-science, and Law. They used PyATE (from spaCy) for extracting the relevant terms from the given corpus and then used Domain Pertinence, Domain Cohesion, and Lexical Cohesion to decide the importance of the term and, then in the second run, they tried NLTK ConsecutiveNPChunkTagger along with it and evaluated the methods based on average precision.

- **Term extraction from domain specific texts** [3]

In this paper, they have used "EUR-lex corpus" made up of financial regulatory documents. After cleaning and tokenization of the data, they formed n-grams (n=1,2,3,4)1 and Supergrams2. Then they scored the terms (n-grams and Supergrams) based on their self-defined metrics which uses C-value, PMI and SWF[3].

1. n-gram length is limited to 4 because they are computationally inexpensive.
2. Supergrams are sequence of validly linked n-grams (after removing overlapping elements).
3. Stop-words fraction, SWF: a measure of noise (#stopwords/#words)

- **Featureless Domain-Specific Term Extraction with Minimal Labelled Data** [4]

  They have introduced a weakly supervised bootstrapping approach using two deep learning classifiers. These two classifiers are trained on a small set of labelled data and then they predict for a subset of an unlabelled data. The strongest predictions are added to the training set and then the classifiers are retrained on this updated training set. This reduces the reliance on labelled data.

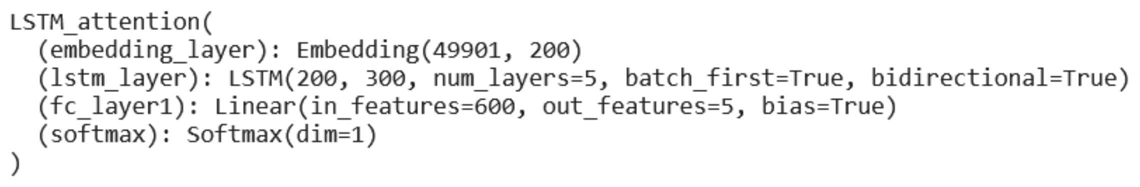
Our Approach

We have extracted the news articles from 5 domains specifically technology, business, sports, politics, entertainment. Then pre-processed the articles and converted the words to word embeddings. Using these word embeddings, trained the classification model to predict the domain of an article. Added the attention layer to check for the terms that have contributed the most to predict the domain of the article. The terms that have the highest attention weights are the most important terms for that prediction and hence these terms are domain-specific keyword. Then prepared the dictionary of these domain-specific keywords.

Dataset

- **Raw-Dataset**
  The dataset for our project contains news articles from multiple domains, including sports, politics, technology, entertainment, and business. We scraped the articles from BBC News, Times of India, and Tech Radar, using a list of URLs as seed links, and collected the text of each article as raw data for our project.
- **Prepossessing**
  The collected data was then cleaned be removing non-alphanumeric values and punctuations for each article and creates two data file one with stopwords used for extracting unigram keywords and other without stop words which was used for extracting bigrams and trigrams
  Finally, the data is stored in a CSV file, with each article represented as a row object. Each row contains the text of the article and its domain.

Architecture



```
LSTM_attention(
  (embedding_layer): Embedding(49901, 200)
  (lstm_layer): LSTM(200, 300, num_layers=5, batch_first=True, bidirectional=True)
  (fc_layer1): Linear(in_features=600, out_features=5, bias=True)
  (softmax): Softmax(dim=1)
)
```

- **Embedding layer** – uses the pretrained glove embedding.
- **Bi-LSTM** – trained the bi-directional LSTM on the output of embedding layer.
- **Attention Layer** – An attention layer is added on top of the Bi-LSTM model by initiating the seed with all zero for generating attention weights.
- **Fully connected layer** – the output of the attention layer is then passed to fully connected layer to output the probability values for all the classes.

Using attention layer, we can find out about the components (i.e., terms of articles that have contributed the most for the correct classification of that article and we can say that term to be domain-specific keyword.

We have tried to find the unigram keywords, bigram keywords and trigram keywords for these 5 domains.

- **Unigram** – Divided the article into unigrams and then using glove embedding, found the embedding for each unigram. After that used these embeddings as pretrained embedding and the attention weights we will get in this case corresponds to each unigram of the article and sorting this value will give the most important unigrams.

- **Bigram** - Divided the article into bigrams and then using glove embedding, found the embedding for each bigram by taking the average of embedding of unigram. After that used these embeddings as pretrained embedding and the attention weights we will get in this case corresponds to each bigram of the article and sorting this value will give the most important bigrams.

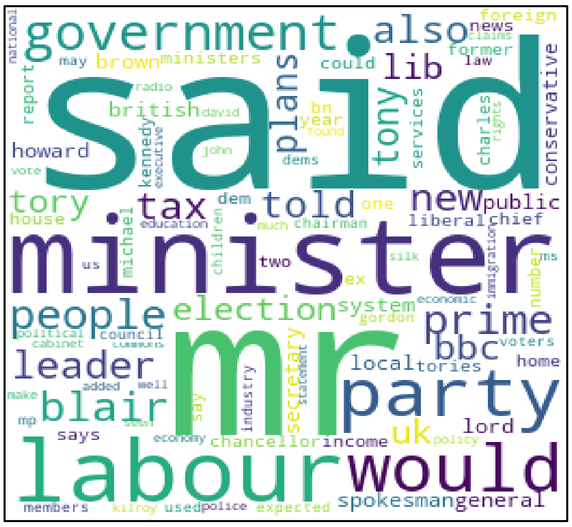
- **Trigram** - Divided the article into trigrams and then using glove embedding, found the embedding for each trigram by taking the average of embedding of unigram. After that used these embeddings as pretrained embedding and the attention weights we will get in this case corresponds to each trigram of the article and sorting this value will give the most important trigrams.

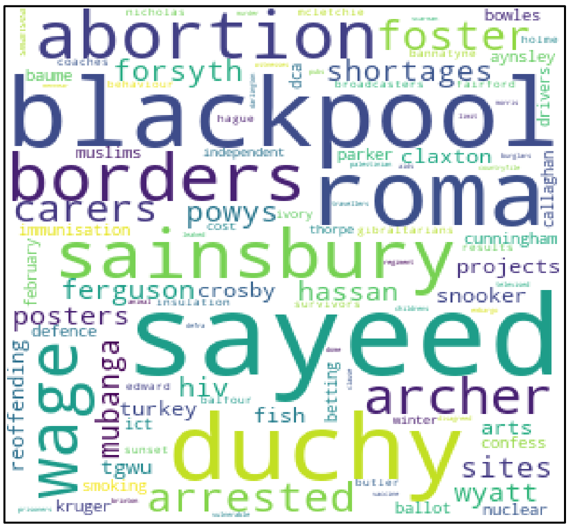
Result

We have observed that the keywords obtained by neural approach are better representing the domain than keywords obtained by tf-idf.

Below is word cloud representation of the keywords with sizes proportional to their normalised ranking among there category

Attention vs Tf-idf unigram word cloud for Politics

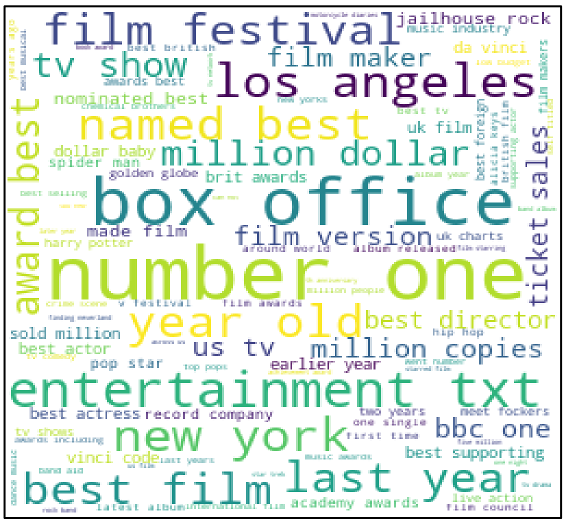


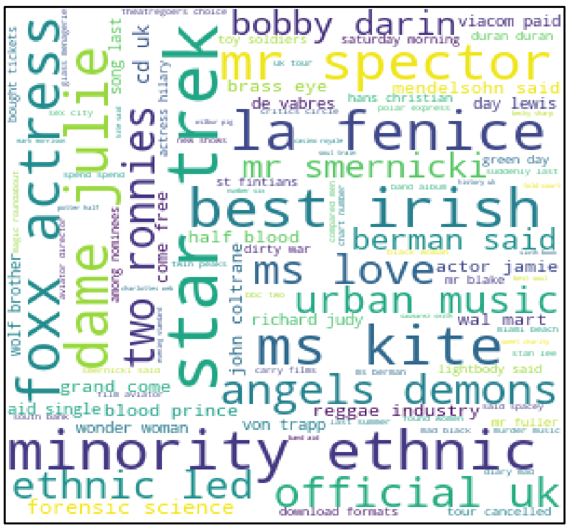


Top10 word

- Using Attention model:
  said, mr, minister, labour, would, government, party, prime, people, blair
- Using TfIdf model:
  fraud, sayeed, blackpool, turkey, students, games, safety, duchy, roma, waiting

Attention vs Tf-idf bigram word cloud for Entertainment



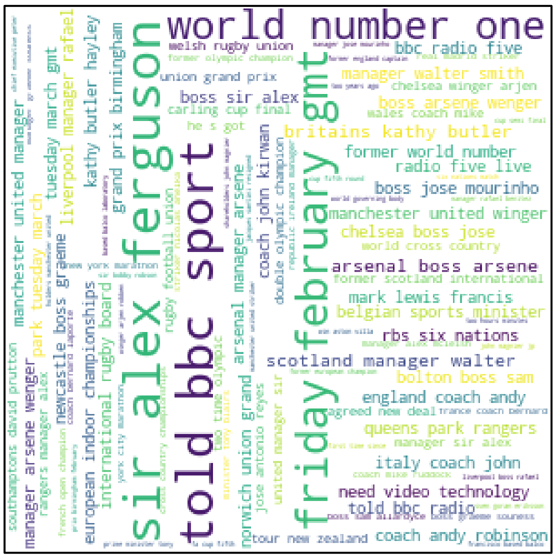


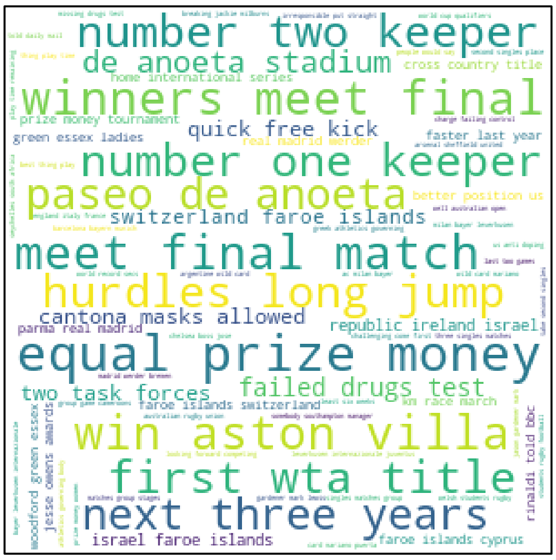
Top10 word

- Using Attention model:
  entertainment txt, number one, box office, los angeles, last year, new york, named best, year old, film festival, best film

- Using TfIdf model:
  star trek, best song, minority ethnic, best irish, ms kite, spend spend, spider man, la fenice, top pops, foxx actress

Attention vs Tf-idf trigram word cloud for Sports



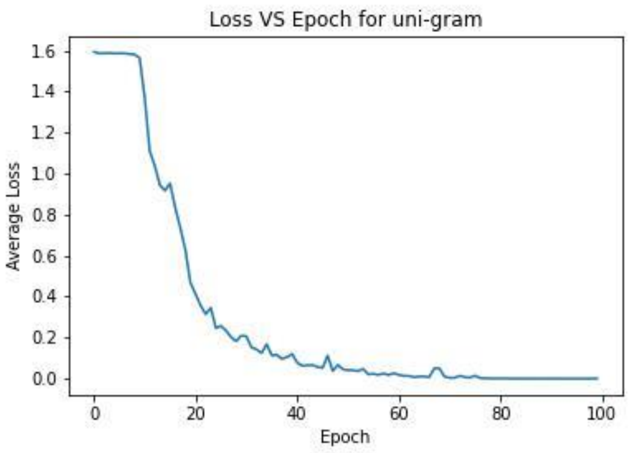


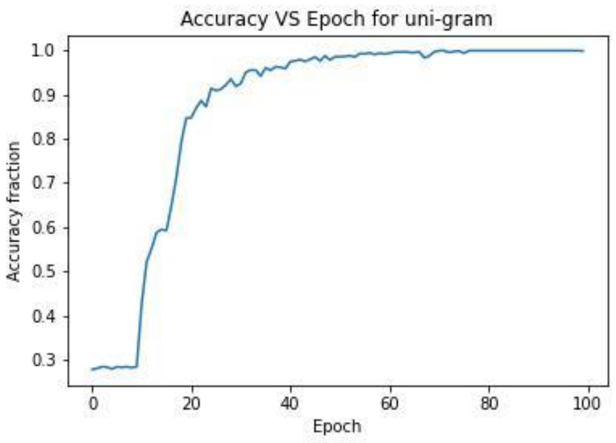
Top10 word

- Using Attention model:
  told bbc sport, coach andy robinson, world number one, sir alex ferguson, rbs six nations, world cross country, bbc radio five, radio five live, european indoor championships, told bbc radio

- Using TfIdf model:
  equal prize money, win aston villa, winners meet final, meet final match, first wta title, number two keeper, number one keeper, hurdles long jump, next three years, paseo de anoeta, de anoeta stadium

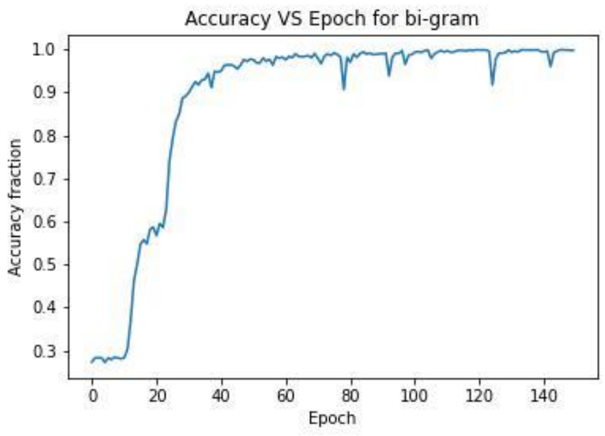
History of our unigram classification model as per its accuracy and loss

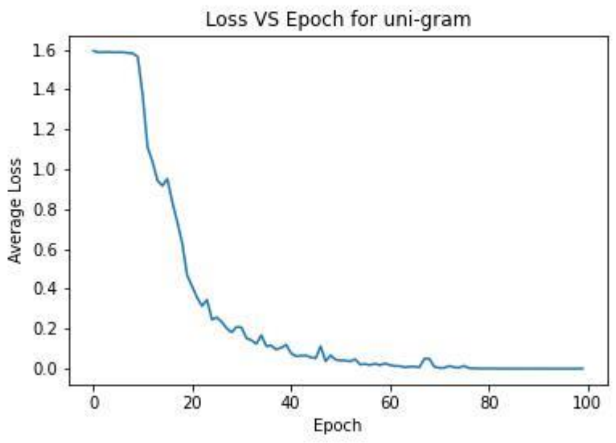




Our model is achieving very high accuracy (approx. 1) after training it for more than 60 epochs. Loss is also decreasing as number of epochs increases. This means that our classification model for unigrams is trained perfectly.

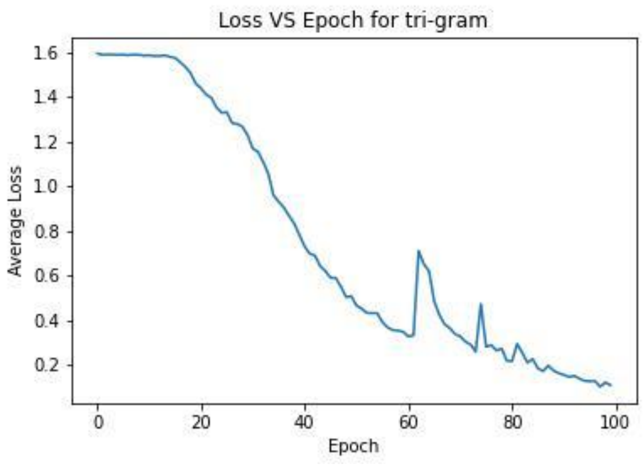
History of our bigram classification model as per its accuracy and loss

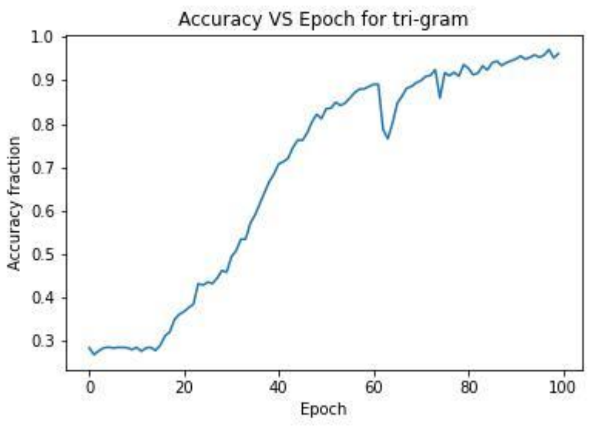




Our model is achieving very high accuracy (approx. 1) after training it for more than 70 epochs. Loss is also decreasing as number of epochs increases. But it is taking a greater number of epochs to achieve the same accuracy as unigram model has achieved in earlier epochs.

History of our trigram classification model as per its accuracy and loss





Our model is achieving very high accuracy (approx. 1) after training it for more than 90 epochs. Loss is also decreasing as number of epochs increases. But it is taking a greater number of epochs to achieve the same accuracy as bigram or unigram model has achieved in earlier epochs.

References

[1] https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiU8cO-yoH2AhXEwTgGHSxpDJEQFnoECAIQAQ&url=https%3A%2F%2Faclanthology.org%2FU09-1013.pdf&usg=AOvVaw341qQBly5U9Qi3JpsIGzwK

[2] https://arxiv.org/pdf/2101.09015.pdf

[3] https://matheo.uliege.be/bitstream/2268.2/7487/7/Master_thesis_2019.pdf

[4] https://aclanthology.org/U16-1011.pdf