Refining Individual's Social Media Content

Sahithya Vemuri **Sriharsha Nakka** **Rakesh Gupta** **Rohith Chodavarapu**
Indian Institute of Technology Guwahati
Group No: 14
{sahit170101077, sriha170101040, sai170101050, rohit170101017}@iitg.ac.in

Abstract

Considering an applicant applying job at a company we develop a NLP language model based method to detect hate speech on online user content and divide the content into different categories that portrays what the post is related to.Through this, user can be made aware of the content they once posted without actually digging deeper into all of their posts and has a choice of removing undesired posts to present himself better. Similarly companies can use some of these tags (other than those used for hate speech detection) to get a gist of applicant's virtual life. We created a corpus of user tweets labelled with many domains. Categories into which posts are classified includes but not limited to Hate, Politics, Entertainment, Food. Then a model is trained to identify abusive language and categories the post fits into through different settings obtain best results. Assuming a list of attributes related to a job description, we compute similarity between the person's tags and job description.

1 Introduction

Owing to enormous use of social media by people, a social media profile has become an identity of an individual itself. With huge emerging job prospects, companies are tending more towards automation of the hiring process by filtering applications based on personality of an individual to check if he/she is the best fit for company in regards of requirements, company's core traits and beliefs. One can say social media of the applicant became a perfect medium to get to know them without actually knowing them.A recent study by the Society For Human Resource Management (SHRM) found that 84% of employers recruit via social media, and 43% of employers screen job candidates through social networks and search engines. Slowly social media is turning from completely personal to more of professional side. So maintaining one's social media profile clean has become very important even from the point of a job application. And one usually has to go through all of his/her social media posts before applying for a job. We aim to solve this problem by developing a method to categorize posts from social media content into various domains including abusive language detection from which he/she can decide how each post affect his/her chances of getting hired by a company and then can refine their profile easily. While refining all kinds of abusive posts is advised, we provide the user more information on that post by post tagging. Also, we intend to develop a supervised text classification methodology which can give a complete outlook of a person along with checking for abusive content. Using this information, we can calculate the probability of an individual to get selected for a job given the list of qualities a company is seeking in their employees. We made a data set of several thousand posts collected from various domains. We have performed several analyses on how models trained on different types and sizes of data across the domains. In subsequent sections data sets used, preprocessing of crude data, arriving at a working model through experimentation over various models, and assessing an individuals suitability for a company are discussed.

2 Method

In this section, we describe the pipeline of our model. In section 2.1, We talk about the creation of our data sets by extracting data from various sources and grouping them. Data cleaning, data preprocessing and data embedding using pre-trained glove model is explained in section 2.2. 3 different models implemented for post tagging are explained in section 2.3. Finally, enumeration of similarity of traits between a job description and an individual is implemented and discussed in Section 2.4.

2.1 Data sets

As we discussed we plan to first give a report to the user saying that he/she has a number of certain negative posts which will give a setback to as almost any company will try to not employ people with negative posts. And secondly, we try to match an individuals posts with specific interests which a certain company is expecting for a role. Since our aim is comprised of 2 objectives, one is to categorize the negative posts and other is to extend this to provide a complete analysis of an individual's profile. We try to collect to 2 types of data, the data to identify and categorize negative texts and the data to tag each text into set of categories which would be helpful to identify a person's interests. We plan to tag the negative comment texts into 6 categories like HATE, OFFENSIVE, TOXIC etc. And the normal texts into a total of 12 categories without the negative categories(e.g. TECH, EDUCATIONAL, BUSINESS etc). Note that we expect to prepare our dataset in which a particular text can be tagged with multiple categories.

For identifying the abusive texts we have corpora (10) with around 150k texts, this data set was taken from twitter and was labelled with 6 categories. We will refer to this as negative dataset for the subsequent sections. Next we want data labelled into categories related to interests with respect to job application. For this type of data we used the corpus from kaggle and can be found (11). This dataset with over 200k texts with headlines and description which was labelled with over 40 categories. We have combined the headline and description into a single text for better context. We have mapped them into 17 relevant categories for job purpose. After that we even cut this data even further into 12 categories dropping the other 6(this include the ones like POLITICS, ENTERTAINMENT etc) to make it more concise. After cutting the data we have collected a total of 85k texts tagged into 12 categories We will call this as positive dataset for the subsequent sections.The graph shown in Figure **??** depicts the data statistics of our positive data set where x and y-axis denotes the categories and frequencies of posts respectively.

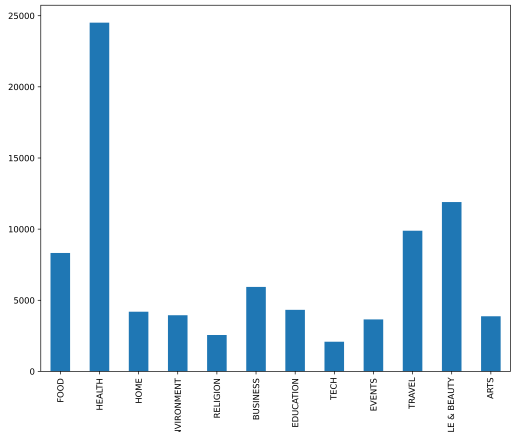

Figure 1: Frequencies of posts vs Categories for negative data set

2.2 Data preprocessing

2.2.1 Data Cleaning

The data we have obtained are from different sources and tend to have various unwanted information depending on the source. In case of tweets, we have removed hastags,@, userid's etc. Later on, in order to consider dataset only in english, we have used wordnet tool from NLTK library to identify non-english words/slang and remove them. Then some stop words along with punctuation marks, multiple spaces and single characters are removed from the data manually. Now we have used regexTokenizer from NLTK library to perform tokenization and lemmatization is carried out by using wordnet tool. Once cleaning is done, empty texts created due to cleaning process are removed. Training, Testing and Validation sets are created by splitting the dataset into 60%, 20%, 20% units respectively.

2.2.2 Text Preprocessing

After we gathered the clean data, we need to have some formatting to feed it to the neural network.This is where Text Preprocessing comes into place. Text preprocessing transforms text into a more digestible form so that machine learning algorithms can perform better.

Methods such as a term- document matrix will result in generation of large vectors for each sentence resulting in unnecessary wastage of memory and resources. Considering this, initially we have fixed our vocabulary size to 5000 words and mapped each word to a number in the range 0-5000 based on its frequency. Extra remaining unique words are mapped to a special number represented as <UNK>. Using this mapping, we have transformed each input text into its corresponding vector but this created a new problem of having varying length inputs to the neural network. To resolve

this problem, we have fixed the length of input to 200 and added padding in case of small inputs or truncated larger ones.

2.2.3 Word Embedding

The transformation of words to vectors currently is purely based on frequency of individual words resulting in no specific relation between similar words/sentences. Due to this, we can assume that any model will fail to understand the relation between the data and ends up giving poor accuracy. Inorder to tackle this problem, we need to transform each sentence with a good relation thus similar sentences/words will get transformed into similar vector/matrixes. An Embedding method is added to facilitate this.

We have used GloVe word embedding for this purpose. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. It does so by leveraging both global and local statistics of a corpus to learn co-occurrence information between each words. Using this knowledge, it creates an embedding matrix to transform each word into its corresponding vector such that similar words will result in similar vectors. Initially,we have used self-trainable models to add embedding but later on we have used pre-trained GloVe models based on twitter and Wikipedia data obtained from here.

2.3 Post tagging

In total, 3 different architectures are implemented for post tagging. As explained in section 2.1, the entire dataset is divided into two categories and two different models are implemented to train on them individually. Finally, we will use both the models for complete tagging using both the trained models. It has been observed the positive dataset is poorly constructed as compared to negative one. As a results, we are able to get good accuracy's on negative datasets using a simple LSTM model implemented in section 2.3.1. But, the results on positive dataset is far worse. So, two other highly complex models are implemented which resulted in a fruitful outcome. The implementation of LSTM model for both datasets is given in section 2.3.1 and the implementation of Bidirectional LSTM model and LSTM with Attention model are implemented on positive dataset in sections 2.3.2 and 2.3.3 respectively.

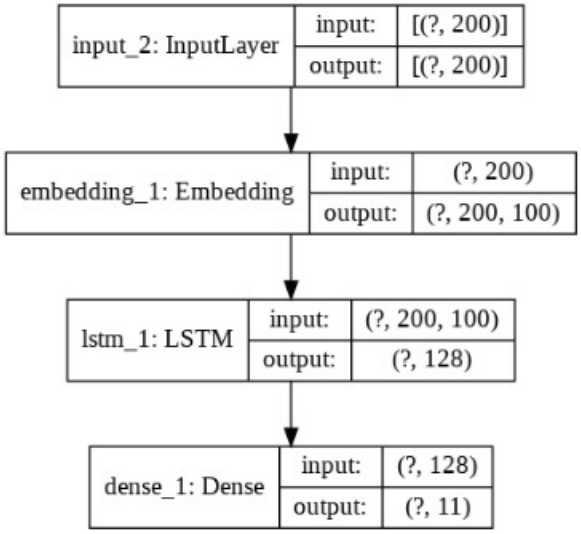

Figure 2: LSTM Model with Glove Embedding

2.3.1 LSTM Model

RNN's are widely used and proved to be efficient in NLP because of their ability to maintain history information which will be very useful to understand context in sentences. Considering the importance of storing long history to understand the context which is essential for tagging, construction of the model is proceeded using LSTM networks. The hidden layer updates present in RNN are replaced by memory cells in LSTM networks making them better at finding and exposing long range dependencies in data which is imperative for sentence structures. A relatively simple model(figure2) consisting of one embedding,one LSTM and one dense layer was implemented for both the datasets. The resultant data after preprocessing and embedding is feeded into these networks.

Cleaned texts after performing text preprocessing are transformed into vector of length 200 each and are fed into embedding layer which is loaded with pre-trained weights obtained from wiki data using GloVe method. Embedding layer converts each word into n-dimensional dense vector(in our case, n=100) satisfying a constraint that similar words should result in similar vectors. Thus, each sentence will be transformed into a 2D matrix. These vectors are fed to LSTM layer with 128 neurons and its output is fed to a fully-connected(Dense) layer to categorize each input into required categories/tags. The texts in positive dataset are categorized into only one category. whereas, the texts in negative dataset are categorized into multiple categories. Both the models

trained on positive and negative datasets have same architecture along with activation,loss functions and other hyper parameters.

Sigmoid function is used as the activation function in the output dense layer. Binary crossentropy is used as the loss function and a batchsize of 128 units is used during the training process. Adam optimizer is used to optimize other hyperparameters of the model. The initial dataset is divided into 80% and 20% datasets for training and testing purposes. Moreover, 20% of the training dataset is used for cross-validation during the training. The entire model is implement using python-keras library and the entire model have around 14million parameters out of which 110k are trainable and the remaining are non-trainable which are loaded from a pre-trained model glove model. Both the datasets are training using this architecture. Due to poor performance observed in positive data model, two other models are implemented in an attempt to increase its accuracy. The comparative analysis of the performance is briefly explained in results section.

2.3.2 Bi-Directional GRU Model

Bidirectional GRU's are a type of bidirectional RNNs with only the input and forget gates. It allows for the use of information from both previous time steps and later time steps to make predictions about the current state. The key difference between a GRU and an LSTM is that a GRU has two gates that is reset and update gates whereas an LSTM has three gates namely input, output and forget gates. GRU use less training parameters and therefore use less memory, execute faster and train faster than LSTM's whereas LSTM is more accurate on dataset using longer sequence. Bi-directional GRUs allows the GRU to learn the problem faster. It is not obvious from looking at the skill of the model at the end of the run. But instead, the skill of the model over time. When all timesteps of the input sequence are available for a problem, Bi-directional GRUs train two instead of one GRUs on the input sequence. The first one will be on the sequence as it is and the second on a copy of the reversed input sequence. This might provide extra context to the network. This can also provide result faster and even full learning of the problem. Our Bi-directional GRU Model consists of 8 hidden layers. After embedding layer we added a dropout layer to prevent overfitting. The output of this layer is connected to bidirectional layer which is in turn

Model: "functional_5"

Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	[(None, 100)]	0	
embedding_5 (Embedding)	(None, 100, 100)	3182100	input_7[0][0]
spatial_dropout1d_4 (SpatialDro	(None, 100, 100)	0	embedding_5[0][0]
bidirectional_4 (Bidirectional)	(None, 100, 256)	176640	spatial_dropout1d_4[0][0]
conv1d_3 (Conv1D)	(None, 98, 64)	49216	bidirectional_4[0][0]
global_average_pooling1d_3 (Glo	(None, 64)	0	conv1d_3[0][0]
global_max_pooling1d_3 (GlobalM	(None, 64)	0	conv1d_3[0][0]
concatenate_3 (Concatenate)	(None, 128)	0	global_average_pooling1d_3[0][0] global_max_pooling1d_3[0][0]
dense_2 (Dense)	(None, 12)	1548	concatenate_3[0][0]

Total params: 3,409,504
Trainable params: 227,404
Non-trainable params: 3,182,100

Figure 3: Bi-directional GRU Model

connected to convolutional layer which allows the network to learn filters. Output of convolutional layer is further fed into global average pooling layer which minimize overfitting by reducing the total number of parameters in the model and into global max pooling layer. The outputs of these layers are then fed into concatenate layer and the output of concatenate layer is finally fed into dense layer to categorize each input into required categories/tags. The overall model is shown in 3.

We have experimented with various activation functions at the output dense layer and with different loss functions. When softmax was used as an activation function at the output dense layer and binary crossentropy as the loss function we got the best results. A Batch size of 128 units is used during the training and adam optimizer is used to optimize other hyper-parameters of the model. We have used this model after reading the article (4).

2.3.3 LSTM Model with Attention Mechanism

When using text extraction we know that some words are more helpful in determining the category of a text than others. However. With LSTM and deep learning methods, while we can take care of the sequence structure, we lose the ability to give higher weight to more important words. We get the best of both words by using Attention.This is thoroughly discussed in the paper (6) and so we based our implementation based on it. Not all words contribute equally to the representation of the sentence meaning. Hence, we introduce attention mechanism to extract such words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector. In essence, we are creating scores for every word in the text or giving different attention to different words. How attention layer sits in the memory and how it affects the output from

a given query is shown in the following Figure 4. To do this, we start with a weight matrix to store

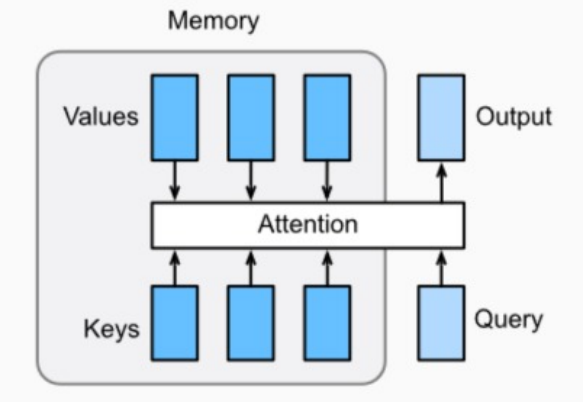

Figure 4: Attention layer interaction

the weights of the words and a context vector. The optimization algorithm learns all of these weights. Intuitively the weight of a word will be higher if the context vector is close to it. After this we send these to the dense layer.

The model we used for LSTM with attention is having 8 hidden layers. After embedding and lstm layer as in section 2.3.1 we added a dropout layer. As we discussed in section 2.3.2 dropout layers help us reduce over fitting so we tried to alternatively add dropout layers in the model. The output of the dropout layer is fed into the attention layer whose output in turn is fed into a dense layer which are followed by dropout,batch normalisation and another dense layer. Softmax function is used as the activation function in the output dense layer. categorical-crossentropy is used as the loss function and a batch sizeof 128 units is fo training. Adam optimizer is used to optimize other hyper-parameters of the model.

2.4 Prediction of suitable company

After detecting the hate speech from social media content of a person, we use the remaining tags to check the key categories under which the content posted falls, to get an insight on his interests, ideology and social behaviour. It is assumed that the companies has a list of attributes to describe the person they are interested to hire. Now using these tags of posts and attributes corresponding to companies, we suggest the person best fit for the applicant. Data in the form of a CSV file obtained from previous phase of the project, which is processed to obtain the categories under which social media content posted by an individual falls under. Also we assumed a list of attributes related to a job description. To achieve this we need to

```
a1=["style","beauty"]
a2=["food"]
a3=["arts"]
a4=["news"]
a5=["food"]
a6=["style","beauty"]
a7=["business"]
a8=["travel"]
a9=["technology"]

b1=["religion"]
b2=["politics"]
b3=["travel"]
b4=b5=b2
b6=["environment"]
b7=["news"]

p1=[a1,a2,a3,a4,a5,a6,a7,a8,a9]
p2=[b1,b2,b3,b4,b5,b6,b7]
```

Figure 5: visualised list of attributes used for persons p1 and p2 obtained from their respective social media content tags ai and bi

find similarity between two lists of words. As tags given by companies may not match with the tags from posts, to get the better of it we can consider the word embeddings or associations for the tags using either GloVe (**?**) or word2vec (8) or fastText (9).Word2vec embeddings captures whether words appear in similar contexts. FastText improves on Word2Vec by taking even the word parts into account. This is desired to get better results for unseen words(not our case). And there is Glove which focuses on words co-occurrences over the whole corpus. Its embeddings relate to the probabilities that two words appear together, which is precisely what we need. So using Glove pre-trained model we obtained embeddings of all the tags and store them in a set. Now the task in hand is to calculate the similarity. Here 2 methods can be used Jaccard similarity and Cosine similarity. As Jaccard similarity takes into account only unique words into account using Cosine similarity gives us better results as it takes into account total length of the vectors. Hence, Cosine similarity is used to get an index corresponding to similarity of traits between

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Figure 6: Formula used to cosine similarity of two vectors

the person and the company. Finally, it is presented in a format that is usable either by the individual or by the company.

3 Individual contribution

Collecting data sets from various relevant sources, data tagging and initial preprocessing is done by Sahithya and Rakesh. Text preprocessing by removing insignificant symbols, hashtags, stop-words, language filtering and tokenization is done Sriharsha and Rohith. Rakesh and Sahithya built the initial model architecture with trainable word embeddings, Rohith and Sriharsha integrated GloVe into the model. At this stage, our initial model is completed. During the second phase, we have made some extra research to discuss the final approaches in both post tagging and company similarity parts. Rohit and Rakesh improved data set creation and implemented the basic two model architecture for training positive and negative datsets. After deciding on using a more complex algorithms, Rohit and Rakesh implement LSTM model with attention mechanism, Sriharsha and Sahithya built bi-directional GRU model. The similairy index between individuals and job descriptions is built by Sriharsha and Sahithya.

Later on, analysing different models with different datasets and fine tuning of parameters are carried out by all of us in order to obtain better results in terms of accuracy.Apart from these, initial research for selecting model algorithm and other work related to formulation of the project idea, creating initial project abstract, midterm report, and final report was contributed equally by all of us.

4 Results & Conclusions

4.1 Results

4.1.1 Post tagging

All the model implemented on google colab platform with 2Gb ram. All these models are trained on the same dataset for post tagging and took around 10 minutes per epoch during the training. The entire training process lasted for around 50 min for each model.

Initially, when entire data is merged into a single dataset, the model accuracy obtained is very inconsistent and it fluctuated around 60%. As discussed in section 2.1, we have later decided to divided our data into negative and positive datasets and performed some category extraction. When these two datasets are trained on two simple LSTM models described in section 2.3.1, we have obtained 99.5% accuracy on negative dataset. But only 29% accuracy is achieved on positive dataset even after 10 epochs and the model saturated around 30. To tackle this problem, two other complex models are implemented on positive dataset. The Bi-Directional GRU Model explained in section 2.3.2 gave 66% accuracy after training for a single epoch and saturated at around 78% after 5 epochs. The LSTM model with Attention mechanism explained in section 2.3.3 also gave similar results with around 80% accuracy after 5 epochs and saturated at the same. A comparative analysis of these results is given in section 4.2. The accuracy vs epoch results obtained for Bi-Directional GRU model and LSTM model with attention mechanism are given in figures x and x respectively.

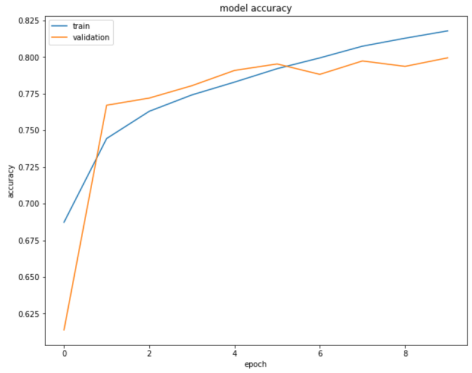

Figure 7: Accuracy vs epoch results obtained with LSTM model with Attention mechanism

4.1.2 Prediction of suitable company

Using the data as shown in 5 and tags given by companies, we are able to provide insights to both the employer and the applicant.

Typical insights provided to the companies are shown in the figure 8. Using these, employer can pick the people who tend to show greater similarity and fulfill the job requirements. Similar insights are shown to the applicants to decide the job they are a perfect fit for.

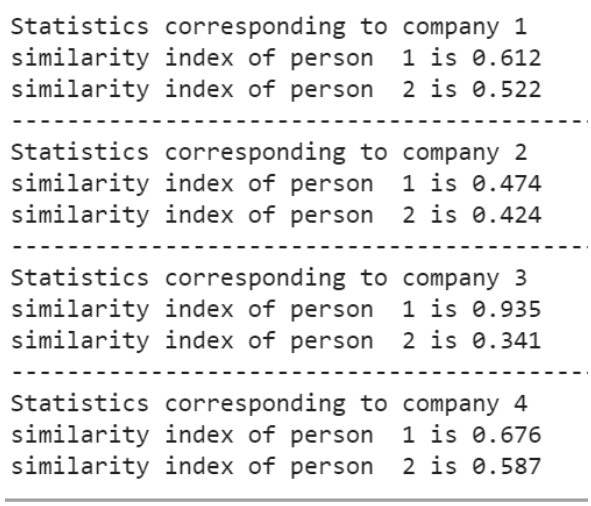

Figure 8: Results obtained for 4 different companies based on the content posted by 2 different persons.

4.2 Conclusions

We have successfully implemented a way to automate the social media refinement process of an individual for job application. We started by creating 2 types of datasets to deal with posts with hate speech and to categorize posts into different tags to identify a person's interests. Then we preprocessed the data to make it suitable for our NN models. Different types of NN models are tried out to get the best accuracy. And using the best model we have made a prediction on individual's posts by finding out the negative posts in the profile and calculating scores for each job role. These scores determine how suitable the person might be for a particular job role. We tried 3 types of NN's as discussed in section 2.3. The folowing table1 shows the accuracy we obtained with all three models.

Table 1: Model accuracy for only positive data

Model	training	validation	testing
LSTM	30.0%	29.6%	29.1%
Bi-Directional GRU	78.4%	78.5%	78.7%
LSTM with Attention	81.3%	79.7%	80.1%

When using the model with only LSTM as in section 2.3.1, Even though the same model is used to train both the datasets, we have observed a huge difference in the performance. This can be considered as a result of improperly built positive dataset and model's incapability to deal with more complex data. We understood that the main problem with LSTM is that it keeps whole text in memory which might not be required for us. So for positive data we used 2 different models one is Bi-directional GRU(section 2.3.2) and LSTM with Attention(section 2.3.3) to solve these problems . Bi-directional GRU implements this by not considering the whole text in the memory while LSTM with attention assigns weights to each word in the text. As we discussed in sectin 4.1.1 with both the models we got a significant improvement in our accuracy. Although LSTM with Attention performed slightly better we think it is specific to our training dataset. Then we used Glove embeddings and cosine similarity to arrive at final presentable results.

5 Limitations and Future works

Our model's accuracy is only 80% for positive dataset whereas for negative dataset it is 99.5%. Initially we collected and used our dataset for which the model gave poor accuracy, so we divided our dataset into two parts and obtained better results for these datasets. From this it can be said that the dataset which we had is a bit poor. Number of categories in to which the posts are classified can be increased to get a precise idea of the gist of the contents of the post which aids in a better similarity index. For positive dataset our two models Bi-directional GRU and LSTM with Attention gave around 80% accuracy. We can try to further improve the accuracy by using a better model. In (5) a new simple model architecture called Transformer is proposed which is solely based on attention mechanisms, dispensing with recurrence and convolutions entirely. This model consumes less time to train and achieves best results. Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data. Instead of assuming that the company gives a set of tags to describe a job, a crawler can be built to automatically pick up the set of key abilities allowing us to further automate the process. As the computation of word embeddings is a little time consuming we can overcome this limitation to some extent by pre computing word embeddings of all tags that are used in classification so that we only need to compute embeddings for tags corresponding to job description. Currently this work is a working prototype it can be further improvised into a web application or mobile application or a cloud service for better usability.

References

[1] Usman Malik. Python for NLP: Word Embeddings for Deep Learning in Keras

[2] Kim Hammar. Deep Text Mining of Instagram Data Without Strong Supervision

[3] Tom Young, Devamanyu Hazarika, Soujanya Poria, Erik Cambria. Recent Trends in Deep Learning Based Natural Language Processing

[4] Felix Silwimba. Bidirectional GRU for Text classification

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin blackTransformed LSTM with Attention

[6] Singh, J.P., Kumar, A., Rana, N.P. blackAttention-Based LSTM Network for Rumor Veracity Estimation of Tweets

[7] Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory, Neural computation, MIT press

[8] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space,2013,arXiv

[9] Bojanowski, Piotr and Grave, Edouard and Joulin, Armand and Mikolov, Tomas. Enriching Word Vectors with Subword Information, 2017

[10] Toxic comments Dataset, Jigsaw Toxic Comment Classification Challenge, 2017

[11] Rishabh Mishra. News category Dataset collected from 2012 to 2018