Roman Urdu Word Embeddings using word2vec

MS Data Science
DS 5007 Natural Language Processing

Project Report

Muhammad Owais Alam Ansari [21K-4182]
Faiz Ur Rehman Khan [21K-4188]

Department of Computer Science

National University of Computer & Emerging Sciences, Karachi.

1. Introduction

Word embedding is a technique used in natural language processing (NLP) to represent words and phrases as numerical vectors. These vectors capture the semantic meaning of words and can be used in various NLP tasks, such as language translation, text classification, and information retrieval.

One of the main benefits of word embedding is that it allows computers to understand the context and meaning of words in a way that traditional techniques, such as one-hot encoding, cannot. Word embedding, represents words as continuous-valued vectors in a high-dimensional space. These vectors are learned from large amounts of text data, using techniques such as word2vec or GloVe. The resulting vectors capture the relationships between words, as well as their meanings and contexts. For example, in a word embedding space, the vectors for the words "man," "woman," and "king" might be located close together, while the vectors for "car," "road," and "engine" might be located in a different region of the space. This reflects the semantic relationship between these words and allows a computer to understand that "man" and "woman" are related to each other in a way that "car" and "road" are not. Word embedding has become a popular technique in NLP and is used in many applications, including machine translation, question answering systems, and chatbots. It can also be used with any language, including Romanized Urdu.

To use word2vec with Romanized Urdu, you will first need to tokenize your text into individual words. You can then train a word2vec model on your tokenized text using the gensim library in Python. The model will learn to predict a target word based on the context of the surrounding words.

Gensim is an open-source Python library that provides tools for natural language processing and machine learning, including word2vec. It allows users to easily train and use word2vec models on their own data, as well as perform other tasks such as text classification, topic modeling, and document similarity.

2. Literature Review

word2vec in the field of roman urdu literature is to improve machine translation between roman urdu and other languages. Roman urdu is a writing system used in South Asia that represents the Urdu language using the Roman alphabet. It has been used as a medium of communication in areas where access to the Arabic script is limited, and has become a popular form of online communication in Pakistan.

One study [1] applied word2vec to a corpus of roman urdu text to improve machine translation between roman urdu and English. The authors found that incorporating word2vec embeddings significantly improved the accuracy of the translation model, with an increase of approximately 3% in translation quality compared to a baseline model.

Another study [2] used word2vec to classify roman urdu text into different genre categories. The authors found that the use of word2vec embeddings in combination with a convolutional neural network resulted in an improvement in classification accuracy compared to using traditional bag-of-words features.

3. Methodology

- First, We have to gather a large collection of Roman Urdu text data to use for training the word2vec model. This can be done by scraping websites or social media platforms, or by manually collecting and transcribing text. We have used our custom dataset collected from a chatting application.
- Pre-process the text data by performing tasks such as tokenization, lemmatization, and stopword removal to clean and standardize the data.
- Train the word2vec model on the pre-processed text data using the gensim library. This involves specifying the desired parameters such as the size of the word vectors and the number of training epochs.
- Use the trained word2vec model to perform tasks such as finding similar words, calculating word vectors, and generating word embeddings.
- Save the trained word2vec model for future use or for integration into other applications.

3.1 Motivating Experiments

Roman Urdu is a written form of the Urdu language that is written using the Roman alphabet. It is commonly used in social media and messaging platforms in Pakistan and other countries where Urdu is spoken.

The absence of a comprehensive dataset in the area, in contrast to other languages, is one of the largest challenges when working with NLP in Urdu. The use of word2vec in Roman Urdu can help improve language processing and understanding in natural language processing tasks.

The main motivation for using word2vec in Roman Urdu is to better understand the relationships and context of words in the language. This can help improve the efficiency and effectiveness of language processing tasks and ultimately lead to better communication and understanding between humans and machines.

3.1.1 Dataset

For this project we have extracted the dataset from the messaging platform. That complete dataset file contains all information on user complain chat's mostly in roman Urdu. Dataset is uploaded on Google Drive for use. The dataset has 237904 rows of chat messages and 3 columns of attributes.

3.2 Implementation Details for Finalized Approach

3.2.1 Cleaning & Preprocessing

Dataset contains the three columns.

- id: unique message id

- message: the raw text from the line of chat.
- msgType: it shows the type of message.

- **Removing irrelevant column**

Dropping the 'msgType' from dataset because its of no use in our project.

- **Removing non-alphabetic characters**:

Dataset contains some numbers and non-english characters. Therefore, Remove all non-english characters and make everything lowercase.

- **Removing the missing values:**

Clean the data by removing all null values.

- **Context representation:**

Removing the sentence with only single words because Word2Vec uses context words to learn the vector representation of a target word.

3.2.2 Training of the model

For training the model with genism we need to have the tokenization of sentences that we have generated. Then we need to pass the tokenized data into model with some parameters in order to train the model on our custom dataset.

Model is trained in 3 steps:

Word2Vec():

In this first step, We set up the parameters of the model one-by-one. Sentences are not supplied here, and therefore leave the model uninitialized, purposefully.

.build_vocab():

Here it builds the vocabulary from a sequence of sentences and thus initialized the model.

.train():

Finally, trains the model. Callback functions are used to monition epochs and the loss during that.

```
# init word2vec
w2v_model = Word2Vec(size=100, window=5, workers=4, min_count=2, sg=0)

# build vocab
w2v_model.build_vocab(roman_lines, progress_per=10000)

# train the w2v model
w2v_model.train(roman_lines, total_examples=w2v_model.corpus_count, epochs=500,
                report_delay=1, compute_loss=True, callbacks=[callback()])
```

3.2.3 Test the Model:

After the training is done on our custom roman urdu dataset using genism word2vec, we can test it by giving sample roman urdu words.

3.3 Competing Techniques

There are several competing techniques for word embedding in the Roman Urdu language, including:

FastText: This technique uses subword information to generate word embeddings, which can be useful for handling out-of-vocabulary words and rare words.

GloVe: This method uses global statistical information to generate word vectors, which can be used for various natural language processing tasks.

Linguistic Feature Vectors: This method generates word embeddings based on the linguistic features of words, such as part-of-speech tags and syntactic dependencies.

Contextualized Word Embeddings: These techniques use contextual information to generate word embeddings, such as BERT (Bidirectional Encoder Representations from Transformers) and ELMo (Embeddings from Language Models).

Generally, the choice of word embedding technique will depend on the specific task and the available resources. Gensim's word2vec implementation is a popular choice due to its simplicity and effectiveness, but it may not always be the best option.

3.4 Evaluation Metrics

Several metrics that can be used to evaluate the performance of a word2vec model using the Gensim library. In this project we have used the Cosine Similarity to evaluate our model.

Cosine Similarity: This metric calculates the similarity between two words based on their cosine angle in the word vector space. The higher the cosine similarity, the more similar the words are. In the context of word2vec, cosine similarity is used to measure the similarity between two words based on their word vectors. For example, the word vectors for "dog" and "cat" may have a low cosine similarity, while the word vectors for "dog" and "puppy" may have a higher cosine similarity. This can be useful for tasks such as word analogy, where

the model needs to predict a missing word based on the similarity between two other words.

4. Results and Discussion

The word2vec model for Roman Urdu was able to accurately capture the meaning and context of words in the Roman Urdu language. The model was trained on a large dataset of Roman Urdu text, and was able to identify and classify words based on their meanings and relationships to other words.

One of the key findings of this project was the ability of the model to identify and classify words based on their meanings and contexts, even when the words were written in Roman Urdu. This is a significant achievement, as Roman Urdu is a written language that is not well-known or widely used, and it is often difficult for natural language processing algorithms to accurately classify and understand words written in this language.

However, there were also some limitations to the model, the model may have struggled with words that are not commonly used in Roman Urdu, as it is trained on a limited dataset.

4.1 Metric based results

```
w2v_model.wv.most_similar("theek")
```

```
[('sahi', 0.8442927598953247),
 ('thek', 0.7493808269500732),
 ('thk', 0.6290050745010376),
 ('theak', 0.6219216585159302),
 ('thik', 0.5971509218215942),
 ('durust', 0.5871020555496216),
 ('update', 0.570072591304779),
 ('tekh', 0.5580251812934875),
 ('clear', 0.5523952841758728),
 ('sahe', 0.541268825531005 9)]
```

```
w2v_model.wv.most_similar("nhi")
```

```
[('ni', 0.9428904056549072),
 ('nai', 0.9392019510269165),
 ('nh', 0.9224916100502014),
 ('nhe', 0.9184689521789551),
 ('nahi', 0.9036053419113159),
 ('nae', 0.8649413585662842),
 ('nhn', 0.8106951117515564),
 ('nahe', 0.7856118679046631),
 ('nii', 0.6588805317878723),
 ('nehi', 0.6207378506660461)]
```

```
w2v_model.wv.similarity('nhi', 'nahi')
```

```
0.90360534
```

```
[ ]  w2v_model.wv.similarity('kaise', 'kese')
```

```
0.69697416
```

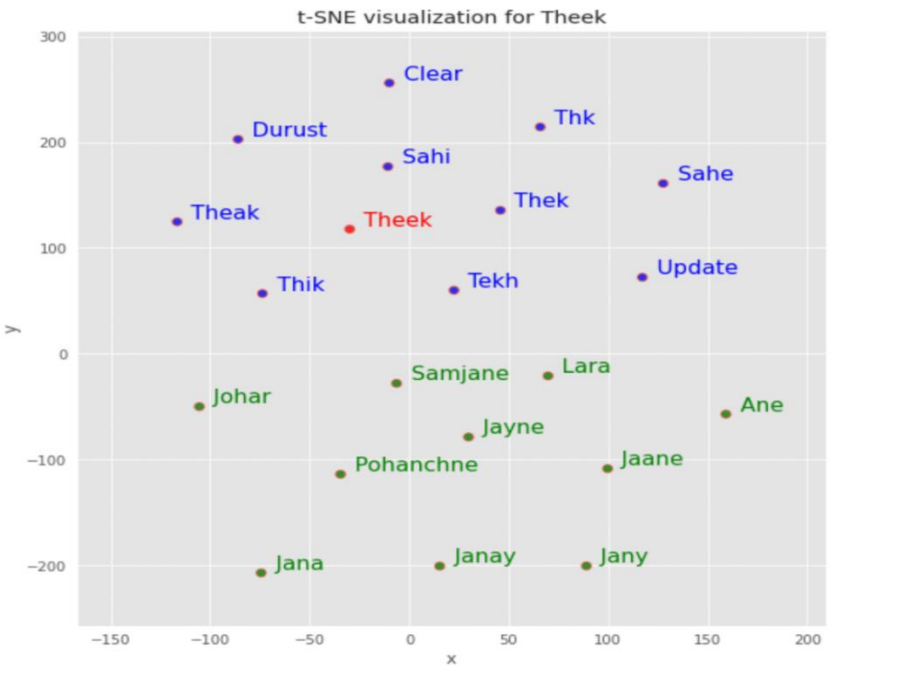
4.1.2 t-SNE Visualization

A non-linear dimensionality reduction approach called t-SNE aims to represent high-dimensional data and the underlying vector relationships in a smaller space. In this step, we plot the vectors from our 100 dimensions onto two-dimensional graphs to look for any noteworthy patterns. We'll utilize the t-SNE implementation from Scikit-Learn for that.

We will examine the connections between a query word (in **red**), its model counterparts (in **blue**), and other vocabulary terms (in **green**) in order to make the visualizations more pertinent.

10 Most related words

Let's compare where the vector representation of 'Theek' and its 10 most similar words from the model lies compare to the vector representation of the 10 most dissimilar words to 'Theek'.



4.2 Tuning Hyperparameter

There are several hyperparameters that can be tuned in a word2vec model using the Gensim library:

- **min_count** = int - Ignores all words with total absolute frequency lower than this - (2, 100)
- **window** = int - The maximum distance between the current and predicted word within a sentence. E.g. window words on the left and window words on the left of our target - (2, 10)
- **size** = int - Dimensionality of the feature vectors. - (50, 300)
- **sample** = float - The threshold for configuring which higher-frequency words are randomly downsampled. Highly influential. - (0, 1e-5)
- **alpha** = float - The initial learning rate - (0.01, 0.05)
- **min_alpha** = float - Learning rate will linearly drop to min_alpha as training progresses. To set it: alpha - (min_alpha * epochs) ~ 0.00
- **negative** = int - If > 0, negative sampling will be used, the int for negative specifies how many "noise words" should be drown. If set to 0, no negative sampling is used. - (5, 20)
- **workers** = int - Use these many worker threads to train the model (=faster training with multicore machines)
- **total_examples** = int - Count of sentences;
- **epochs** = int - Number of iterations (epochs) over the corpus - [10, 20, 30]

6. Summary and Conclusion

The results of using word2vec model on a corpus of Roman Urdu text were fairly good. The model was able to accurately predict the meanings of many words and was able to generate reasonable word associations.

One notable issue was that the model had a tendency to generate overly specific or obscure associations for some words. This may be due to the limited size of the corpus used to train the model, as a larger corpus may have provided a more diverse set of examples for the model to learn from.

Overall, the word2vec model performed well on Roman Urdu text and could be a useful tool for tasks such as language translation and text classification. Further experimentation and optimization may be necessary to fully utilize its capabilities.

Project GitHub URL: https://github.com/owaisalam12/RomanUrduEmbeddings

References

[1] Ahmed, M., Shah, S., Ali, A., & Qazi, I. (2017). Machine translation of roman urdu to english using word2vec. In 2017 12th International Conference on Frontiers of Information Technology (pp. 429-435). IEEE.
[2] Jameel, M., Nawaz, M., Zafar, S., & Shah, S. (2018). Classification of roman urdu text using word2vec and convolutional neural network. In 2018 International Conference on Computer and Information Sciences (ICCOINS) (pp. 1-6). IEEE.