Relation Extraction in Knowledge Graphs using Deep Semisupervision

Kartik
kartik@ucsc.edu

Abstract

With the rising use of virtual assistants and conversation agents, it is imperative to build systems that can automatically understand the meaning, intent and relations among the entities in the text. In this work, I propose a novel score-fusion based deep sequence-to-sequence method for extracting knowledge-graph relations from utterances provided to conversational systems. I evaluate the approach with the various deep learning algorithms and other techniques.

1 Introduction

Natural language Understanding (NLU) is central part for fulfilment of any task in conversation agents such as online help/chat bots, assistants in driverless vehicles and virtual assistants. This involves understanding hierarchical semantic frame and retrieving relevant information i.e. domain, intent, slots and relations. Knowledge Graph (KG) enables an abstract and convenient way to organize large amount of information on the web in the form of nodes and edges. The representations learned from KGs are being used to solve a wide variety of downstream tasks such as question answering, link prediction, entity classification and information retrieval. Fig.1 shows an example of a knowledge graph from the few movie samples in the dataset.

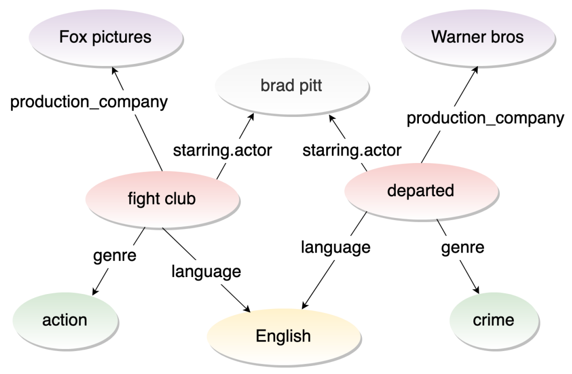

Figure 1: Knowledge graph for movies

Relation extraction is the task of extracting all the relations from an utterance. Fig. 2 shows few examples of an input utterance, its knowledge graph fragment, and the extracted relation.

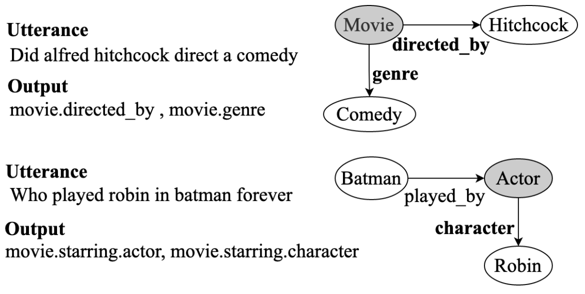
Utterance
Did alfred hitchcock direct a comedy
Output
movie.directed_by , movie.genre
Movie
Hitchcock
directed_by
genre
Comedy
Utterance
Who played robin in batman forever
Output
movie.starring.actor, movie.starring.character
Batman
Actor
played_by
character
Robin

Figure 2: Relation Extraction from utterance

In this work, I provide a comprehensive comparison of conventional machine learning algorithms on a database $\mathcal{D}$ of utterances addressed to conversational system. The first section talks about the analysis of the dataset. Next, I formally define the task and discuss the methodologies used for relation extraction. Further, the experimental results are discussed.

2 Dataset

The Dataset consists of 3234 sentences with utterances including movie names, movie genres, actors, directors, producers, release dates, languages etc. The task is to extract relations from the utterances where each utterance can have multiple possible classes. The train and test set consists of 2253 and 981 samples respectively and a total of 19 classes.

As any real-world dataset, this dataset is highly imbalanced with the class `movie.starring.actor` having highest frequency of 340 samples and `movie.locations` with lowest count of only 3 samples. Fig.3 shows the frequency distribution of all classes in the dataset. Further, the different distribution of words in train and test poses another challenge as 35.7% words in test set are not in present in the train set.

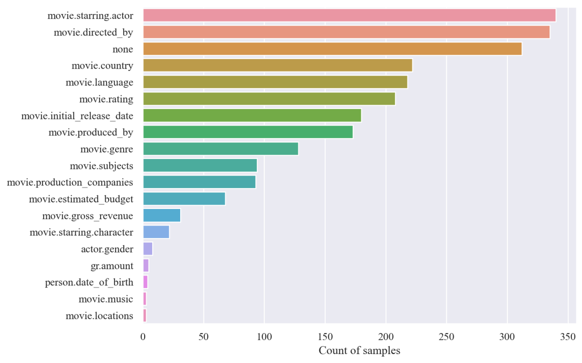

Figure 3: Frequency Distribution of classes in D

Further, I analyze the dataset for the popularity of words. Fig. 4 shows the wordcloud of the entire text where the fontsize represents the frequency of a word. Unsurprisingly, I observe that the word *movie* is the most popular word in the dataset followed by *show*, *produced*, *director* and *film* etc.

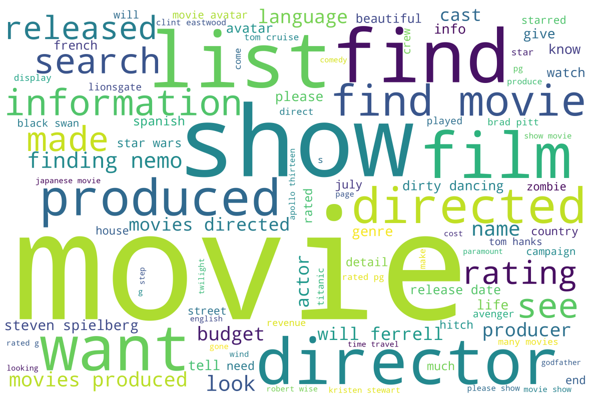

Figure 4: Wordcloud of the Dataset

3 Methodology

This section describes the process of extracting features from the sentences and methodology adopted for the multilabel classification task.
Task Definition: *Given a labelled training dataset* D *with a set of utterances* $\{u_1, u_2, \ldots, u_n\}$ *, the task is to learn a prediction/classification function for an unknown utterance* x *that predicts if a label* $l_i \in L$ *is present in it or not, where* L *is a set of possible relation labels. Note that a single utterance can have multiple possible labels.*

3.1 Feature Extraction

Representation of input textual data into meaningful features is a crucial component of any successful machine learning algorithm. The following features are used in the experiments:

1. *Count Vectors:* One of most popular and basic text feature methods is count vector. Count vectors just forms a $n \times n$ matrix of word counts for words in the corpus.

2. *Term-Frequency Inverse-Term Frequency (Tf-idf) vectors:* TFIDF calculates the importance of word using its frequency in documents. Most occurring words get lower importance and vice versa.

3. *Glove Embeddings:*(Pennington et al., 2014) These embeddings are obtained by unsupervised learning of word-word co-occurrence on large data. Words close together in the semantic space have similar glove embeddings. For a sentence, embeddings are calculated by averaging the embedding of each word in that sentence.

4. *Fast-text embeddings:*(Bojanowski et al., 2017) The novelty of fasttext is to use the internal structure of a word to improve vector representations obtained from the skip-gram method. This allows training word embeddings from a training corpus with the additional ability to obtain word vectors for out-of-vocabulary words.

3.2 Methodology

For the best performing model, I use a mix of supervised fusion and unsupervised pos-tagging algorithm to train the model. This is based on the following observations:

1. It becomes almost impossible for the model to classify a proper noun to its class without knowing what it represents. For example: Consider two sentences a) *quentin tarrantino movies* and b) *bette midler movies*. a) is labelled as movie.directed-by but b) is labelled as movie.starring.actor. Without having external knowledge about their professions, even humans can't answer it correctly. Problem becomes worse, when the noun has very few occurrences in the dataset, making it difficult for model to relate noun with a class. For this, I use noun chunking combined with external IMDB dataset[1] for mapping nouns to a key. Any occurrence of that noun in the dataset is replaced by key+noun.5

[1]https://www.imdb.com/interfaces/

2. Every model after training gets skewed towards the class with majority samples. This problem amplifies with classes less than 20 samples since the model has very limited data points to learn useful information about that class.

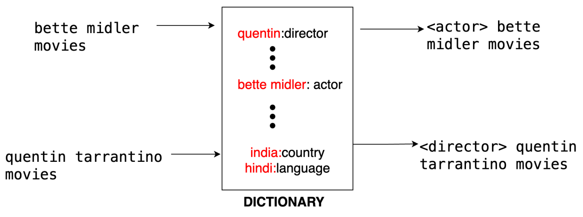

Figure 5: Dictionary based approach

4 Experiments

4.1 Baseline Methods

I consider 5 conventional ML models for preparing baselines on the dataset.

1. Multilayer Perceptron (MLP): MLP is a feed-forward neural network which consists of fully-connected linear layers. This means that all the neurons in the input are connected with all the neurons in the next layer. Cross-entropy is used to calculate the loss and gradient descent algorithm is used for updating the parameters (weight and biases).

2. Recurrent Neural Networks (RNN) (Zaremba et al., 2014): Instead of passing input as a single linear layer in case of a neural net, RNN allows runs in a sequence-to-sequence manner which takes the benefit of previous dependencies in order to predict a class. Each input needs to be in a form of seq-length*embedding-size to capture previous dependencies.

3. Long Short Term Memory (LSTM)(Hochreiter and Schmidhuber, 1997): LSTMs are a class of RNN as they also use a chain-like structure for processing a sequence. However, in order to retain long-term dependencies, LSTM use a gating-mechanism to remember useful information and forget useless ones. It consists of 3 gates: input gate, forget gate, output gate. The input format is the same as an RNN.

4. Convolution Neural Network (CNN) (Cortes and Vapnik, 1995): CNN are similar to an MLP but they use spatial information to calculate the importance of features in an input. A fixed size kernel convolves over the text/image to calculate the positional features. Different size kernels are useful for finding different types of feature information.

5. Score Fusion (SF): Different algorithms capture different useful information from the data. Also, a single model giving maximum confidence of a certain class among all the classes can have a low confidence level on that prediction. For example: Confidence 0.55 vs 0.9 both give True. Using multiple models in combination can normalize the confidence score if both models are not sure about a prediction. I use score fusion, i.e. average out the probability scores from 2 models to select the max confidence class.

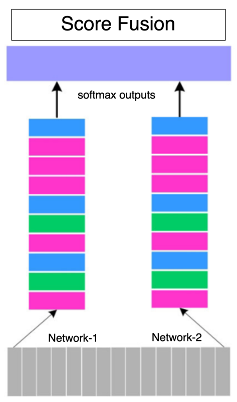

Figure 6: Score Fusion technique for robust predictions

4.2 Implementation Details

Algorithms: All algorithms are implemented using Python3.7 and open-source Pytorch [2] (Pedregosa et al., 2011) version 1.9. Pretrained-embeddings are initialized using gensim(Řehřek et al., 2011).

Parameter Settings: All experiments are performed on 80% train set and hyperparameters are tuned using manual search on the remaining 20% data as validation set. The random seed is set to 1. Table 1 shows the best parameter values of all experiments.

5 Results

In order to evaluate the performance of the models, we use *Accuracy* as the metric. In a multiclass

[2]https://pytorch.org/

Model	Parameters
MLP	n-layers:2, input-dim: vocab-size criterion:cross-entropy, n-epochs:25, learning-rate:0.001
RNN	n-layers:1, embedding-dim:500, hidden-size:256, criterion:cross-entropy, n-epochs:15, learning-rate:0.0001
LSTM	n-layers:1, input-dim:500 criterion:cross-entropy, n-epochs:15, learning-rate:0.001
CNN	n-kernels:4, input-dim: 500 criterion:cross-entropy, n-epochs:5, learning-rate:0.001, dropout:0.25

Table 1: Hyperparameters of the tuned-models

multilabel problem, a prediction is correct if all the labels are correctly identified for an utterance.

Table 2 shows the results when solved as a multiclass multilabel problem for each class. Labels are converted to One-Hot Vectors (OHV) where each column belongs to a single class i.e. 19 columns. Table X shows the performance of all models on the training and validation sets. MLP networks gives the best performance with a comparable performance as an LSTM.

Model	Train Acc.	Val Acc.
NN+Tfidf	97.1	81.5
RNN	90.1	78.7
LSTM	92.8	81.3
CNN	97.4	75.5

Table 2: Performance of Multilabel Multiclass Classification models

Next, I approach the problem solely as a multilabel classification problem where the model predicts 1 class for an utterance. This was based on the assumption that the test set has the exact same combination of classes as the train set. Multiple classes in an utterance are considered as a single class which converts the 19 classes into 47 classes. Classes with less than 10 samples are discarded are they act as noise in the dataset and models are trained on the remaining class. Table 3 shows the comparison of various models. Similar to previous results, MLP again gives better performance despite its simplicity due to the relative less number of parameters involved. This seems reasonable given the less amount of training data available. LSTM however gives a little less performance as very long term dependencies are not required. Average sentence length in the training data is 6 words only.

Model	Train Acc.	Val Acc.
NN+Tfidf	99.5	89.8
RNN	93.1	79.5
LSTM	97.8	87.3
CNN	99.4	88.0
Fusion	99.1	90.2

Table 3: Performance of Multilabel Classification models

To improve upon the validation accuracy, I use a score fusion technique to increase the confidence score among the classes. The final softmax layer from two networks are extracted and scores are averaged out. Next the maximum confidence class is chosen to remove out the low-confidence classes. This improved the validation score by 1% accuracy.

6 Conclusion

Relation Extraction is a challenging problem given the limitations of traditional machine learning models and limited availability of annotated data. Skewed class distribution poses another problem as it induces bias while training. A combination of supervised and unsupervised external-knowledge based approach is proposed in the work to solve the task of relation extraction. Future directions include use of deep-learning based approaches to handle the multiclass multilabel problem without splitting into multiple models.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Radim Řehřek, Petr Sojka, et al. 2011. Gensim—statistical semantics in python. *Retrieved from genism. org*.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.