Relation Extraction from Natural Language using PyTorch

Shashwat Pandey
University of California, Santa Cruz / Santa Clara, CA
spandey7@ucsc.edu

1 Introduction

The goal of this homework is to develop deep learning models to identify knowledge graph relations that are referred to in user utterances to a conversational system (in this example, according to the Freebase schema). In this report we will be going through the entire journey of development and analysis to finally conclude with the experiment results and conclusion.

1.1 Problem Statement Analysis

For a given dataset of user utterances and core relations, we have to create a multi-label perceptron and predict relations on test data of user utterances. After generating predictions in csv format, we have to use evaluation metrics to figure out how our model is doing on untouched data.

Through the problem statement, we can deduce that we are supposed to carry out a classification based supervised learning. The data is supposed to be interpreted in text format making it a Natural Language Processing(NLP) based problem.

We have a limited set of categorical values that will be used as target labels, over which we will train user utterances and learn the relationship of each utterance with the label.

To begin with, we'll be using the CRoss Industry Standard Process for Data Mining (CRISP-DM) pipeline for our analysis and modelling, which is the de-facto standard and an industry-independent process model for applying machine learning projects.(1). Figure 1 provides an exact idea of the flow supposed to be followed before tackling any data-science/machine learning based problem and its entire lifecycle.

The following section talks about the analysis on the dataset

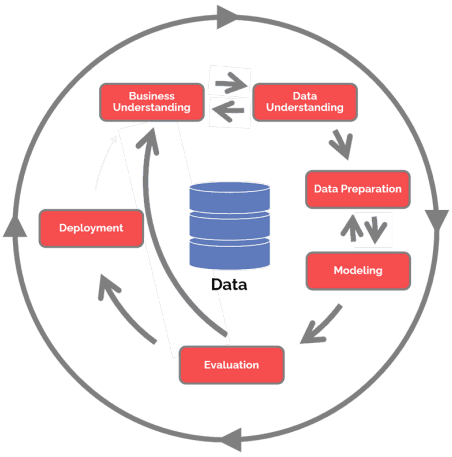

Figure 1: CRISP-DM Diagram: Inspired by Wikipedia

1.2 Dataset

The training dataset consists of 2312 user utterances specific to films and their specifics. The test dataset on the other hand, consists of 980 utterances for which we have to predict the core relations. Along with that, we are provided with their core relations that give us an insight on what each utterance requires. Each of these sentences are associated with either one or greater core relations. Being a supervised learning problem, "UTTERANCES" will be considered as inputs and "CORE RELATIONS" will be taken as target labels/output.

As an example given an utterance, "find the female actress from the movie she's the man", the output label will be "movie.starring.actor", and "actor.gender". In total we have 19 such unique labels.

Going ahead with data exploration, we'll try to find some more information on the data through visualisations, and do the required pre-processing and feature extraction before we design a model and train the dataset on it.

1.3 Visualisations

Getting a visual understanding of the dataset is essential for any data analysis. With our given data of user utterances and relationship labels, we have tried to learn more about the data through its vocabulary, word frequency and label balance.

We examine the dataset to determine which words are popular. The full text is represented as a wordcloud in Figure 2, where the size of the typeface indicates how frequently a word appears. However, we can see the frequency of words such as ”the”, ”in”, ”of” overpowers vocabulary essential to our training. Hence, we filter the stopwords out of the vocabulary and create another word cloud shown in Figure 3. It comes as no surprise to that the term ”movies” is the most common one in the dataset, followed by the words ”show”, ”find”, ”directed”, and ”director”, among others.

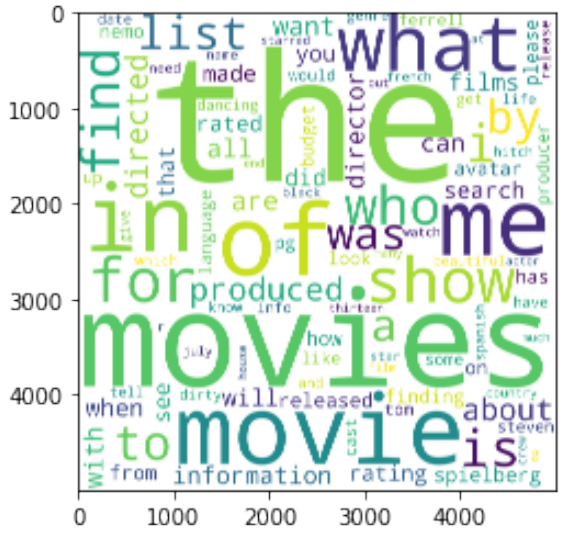

Figure 2: Word Cloud of top 100 most used words in utterances before removing stopwords

This dataset, like any real-world dataset, is quite unbalanced, with movie.directed.by having the highest frequency (323 samples), and movie.locations having the lowest frequency (just 3 samples). The frequency distribution of top 13 classes in the dataset is shown in Figure 4.

2 Methodology

2.1 Feature Extraction

Any effective machine learning algorithm must successfully represent the incoming textual data as meaningful features. We have made use of the following tools in our experiment:

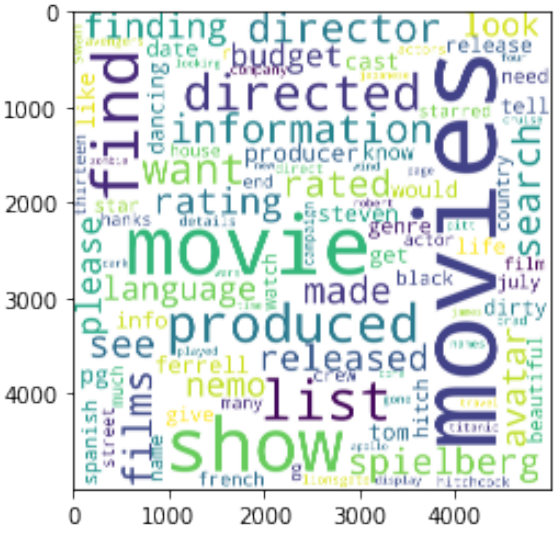

Figure 3: Word Cloud of top 100 most used words in utterances after removing stopwords

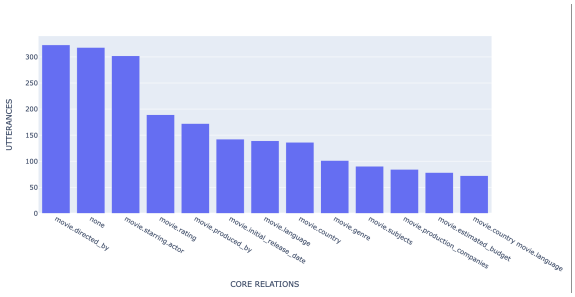

Figure 4: Bar graph of words labels having ¿50 frequency

1. *Vectorization*: It is the most widely used and fundamental text feature technique. Vectorizations is simply forming a count matrix of *n* x *n* dimensions for the words in the corpus. It is essential to almost every NLP based model out in the world as computations can only be done on numeric data and not string based data.

2. *Term Frequency - Inverse Document Frequency (TFIDF)*: TFIDF uses a word’s frequency in documents to determine how important it is. Most frequently occurring terms are given less weight, and vice versa. It was essential specifically for the given dataset and problem statement as we had the data of different movies, actors, directors being used only once or twice in the entire list of more than 2000 utterances.

2.2 Model Implementation

Recent studies have demonstrated the effectiveness of the multilayer perceptron, as an alternative to more conventional statistical methods. The multilayer perceptron(MLP) can be trained to roughly resemble any smooth, measurable function, as has been demonstrated in many research endeavours of late. The MLP, in contrast to other statistical methods, makes no prior assumptions about the distribution of the data. When given fresh, untested data, it may be trained to generalize with accuracy and represent extremely non-linear functions.(2) It is because of these characteristics, MLP is considered as a compelling alternative to creating numerical models and will be used by us for this experiment.

The MLP is made up of a network of straightforward, interconnected neurons, or nodes. The weights and output signals connecting the nodes are a function of the total of the node's inputs, as adjusted by a straightforward nonlinear transfer, or activation, function.

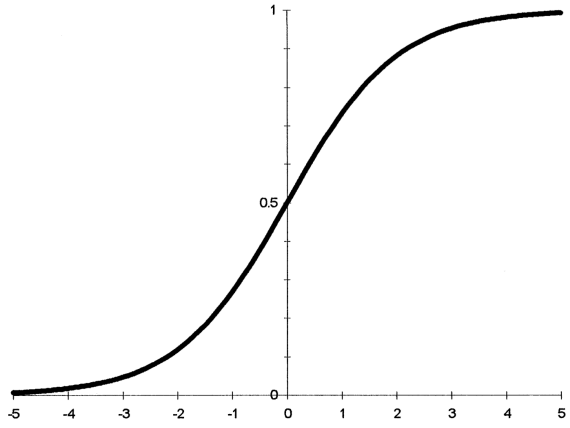

Figure 5: The logistic function y = 1/(1 - exp(-x))

The MLP can estimate straightforward nonlinear processes by superimposing numerous small nonlinear transfer functions. It could only model linear functions if the transfer function was linear. The logistic function, as shown in Figure 4, is a frequently utilized transfer function because of its easily determined derivative. The connecting weight scales a node's output before feeding it forward as an input to the nodes in the following layer of the network. Because of this, MLP is also known as a feed-forward neural network. This implies a direction of information processing.

MLP can have a variety of architectural configurations, although in most cases it will include multiple layers of neurons. The input layer only transfers the input vector to the network; it does not perform any computations. The inputs and outputs of the model are referred to as input and output vectors, which can be represented as single vectors as seen in Figure 5. An output layer comes last in the MLP model, which may also have one or more hidden layers.

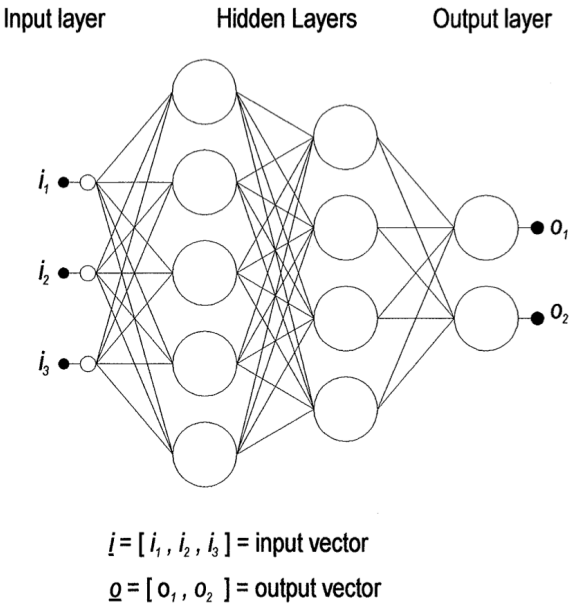

Figure 6: A multi-layer perceptron with 2 hidden layers

Each node in an MLP is said to be fully connected to every other node in the layer above and below. It has been demonstrated that a MLP can approximate any smooth, measurable function between the input and output vectors by choosing an appropriate collection of linking weights and transfer functions.(3)

Multilayer perceptrons can acquire new skills through practice. The training data is repeatedly fed into the perceptron, and the weights in the network are changed until the desired input—output mapping is achieved. These are mostly used for supervised learning models. For a specific input vector, the MLP's output during training may not match the expected output. The disparity between the desired and actual output is referred to as an error signal. The amount of this error signal is used during training to decide how much the weights in the network should be changed in order to lower the model's overall error. MLP can be trained using a variety of approaches. After being trained using training data that is sufficiently representative, it can generalize to fresh, untested input data.

On a basic single or double layered perceptron,

(the one which will be used by us), not many hyperparameters are offered to tune and converge the loss to a global minima. We are supposed to feed the learning rate, epochs and batch size of the training set used by the model. If we want to increase the training set for better learning, we can also avoid splitting the data into training and validation set, although it is not recommended as it could lead to overfitting. Furthermore, we can also dabble with number of hidden layers and add/remove some to observe changes in the loss and get a better accuracy over the model.

3 Experiment

3.1 Train-Test Split

With an 80:20 ratio, the data set was divided into a train and validation set. As a result, the training dataset includes 462 data samples to assess the model's correctness in addition to 1850 data samples for training. The dataset is shuffled before splitting in order to provide a small sample of data for each different category in both training and validation. We have used the 'train-test-split' function of Scikit-learn(4) to divide the data. Since the function train-test-split shuffles the input each time and produces a different set of results, random state was enabled.

3.2 Hyperparamter Tuning

All 19 of the original labels were considered as outputs in this method. With OneVsRest, the challenge of multiclass/multilabel classification is divided into numerous binary classification problems by fitting each class against all the other classes for each classifier.

While training the dataset with different hyperparameters, we also tried various data cleaning techniques as well. Going forward, we first ran the model after removing all the stopwords from the dataset and saw a dramatic drop in the validation and test score. Along with this we tried different combinations with loss functions to get the best possible accuracy out of the model. The following tables show how the model performed with different loss functions, hyperparameters and stopwords.

Loss*	**LR**	**Epochs**	**Batch**	**Val Acc**
CE	0.01	50	4	0.66
CE	0.01	50	16	0.58
CE	0.1	100	4	0.73
CE	0.01	100	16	0.54
CE	0.0001	20	4	0.74
CE	0.0001	20	16	0.70
BCE	0.01	50	4	0.64
BCE	0.01	50	16	0.52
BCE	0.1	100	4	0.71
BCE	0.01	100	16	0.64
BCE	0.0001	20	4	0.72
BCE	0.0001	20	16	0.69
BCE-Logits	0.01	50	4	0.68
BCE-Logits	0.01	50	16	0.59
BCE-Logits	0.1	100	4	0.71
BCE-Logits	0.01	100	16	0.61
BCE-Logits	0.0001	20	4	0.71
BCE-Logits	0.0001	20	16	0.68

Loss	**LR**	**Epochs**	**Batch**	**Val Acc**
CE	0.01	50	4	0.81
CE	0.01	50	16	0.73
CE	0.1	100	4	0.72
CE	0.01	100	16	0.70
CE	**0.01**	**100**	**4**	**0.90**
CE	0.0001	20	4	0.81
CE	0.0001	20	16	0.80
BCE	0.01	50	4	0.78
BCE	0.01	50	16	0.69
BCE	0.1	100	4	0.88
BCE	0.01	100	16	0.80
BCE	0.0001	20	4	0.85
BCE	0.0001	20	16	0.80
BCE-Logits	0.01	50	4	0.76
BCE-Logits	0.01	50	16	0.70
BCE-Logits	0.1	100	4	0.82
BCE-Logits	0.01	100	16	0.75
BCE-Logits	0.0001	20	4	0.81
BCE-Logits	0.0001	20	16	0.78

Using sckit-learn's tfidf function, input utterances were transformed into word feature vectors. We then used the multi-label-binarizer in scikit-learn to translate labels into one hot encoded vector. Following this , a two-layered perceptron neural network was given the tfidf feature vectors. Considering that the issue necessitates a number of neurons in the final layer, we used Pytorch's Cross Entropy as the loss function. We then passed the tensors through a sigmoid function to get the output

layer in its desired form. Multi-class issues are better suited to the sigmoid function. For tuning the hyper parameters a variety of values were tried for the learning rate, batch size, epoch, optimizer function and loss function.

In the tables above, the first table with "Loss*" has been trained without stopwords and the latter without the asterick has been trained with stopwords. The model and hyperparameters that gave us the best accuracy out of all can be identified in bold font. Of course, many more iterations of training were run, but for the sake of the space all couldn't be included here.

3.3 Evaluation

Based on the results of the confusion matrix, the classification report, and the validation accuracy score, the model's performance is assessed. The examples in each actual class are represented by each row, while those in each anticipated class are represented by each column, or vice versa, in the confusion matrix. However, the classification report calculates each model's precision, recall, and f1 score. Precision is a ratio of True Positives (TP) to total positives guessed (TP + FP). Recall is the ratio of True Positives to actual positives (TP + FN).

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2PR}{P + R}$$

4 Results

After running multiple training and validation iterations, we came to the conclusion that the model was able to train better with the stopwords. Our observation stems from the fact that stopwords provided emphasis on certain nouns and verbs to help the model understand the relationship between the utterances and labels better.

Figure 7 and Figure 8 give us a visual representation of how well the best model was performing on training and validation sets. As you can see in the loss curve, we have tried to avoid overfitting as much as possible. The two-layered multilayer perceptron provided us with the best test accuracy under the used hyperparameters. Due to the constraints of timeline and limits on submission, further feature extraction techniques such as lemmetization, POS tagging couldn't be explored.

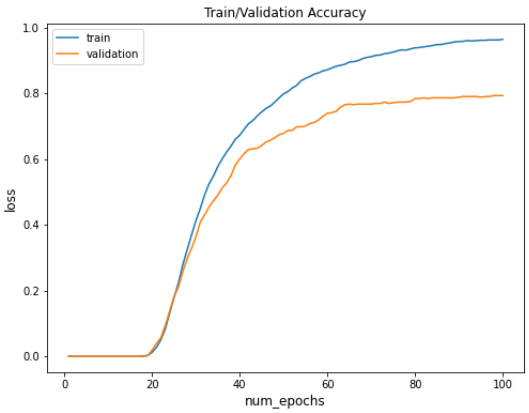

Figure 7: Training-Validation Accuracy

Given the limits of conventional machine learning models and the scarcity of labeled data, Relation Extraction is a difficult topic. Another issue is that a biased training environment is created by a skewed class distribution. In order to complete the task of relation extraction, a combination of supervised and unsupervised external-knowledge based approaches is suggested. The employment of deep-learning based methods to address the multiclass multilabel problem without breaking into numerous models is one of the areas we would like to look into in future.

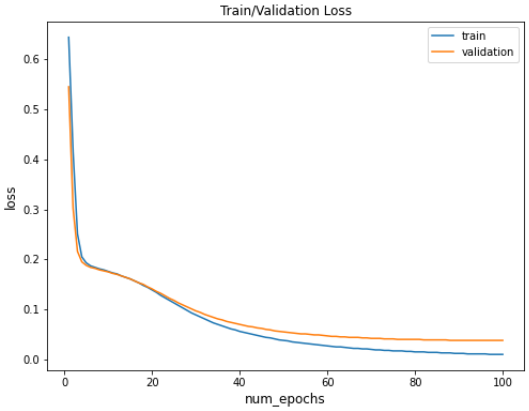

Figure 8: Training-Validation Loss

Model	Train	Validation	Test
2-Layered M-labelled	0.96	0.89	0.77

References

[1] Schröer, C., Kruse, F., Gómez, J. M. (2021). A systematic literature review on applying CRISP-DM process model.

[2] Gardner, M. W., Dorling, S. R. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric environment, 32(14-15), 2627-2636.

[3] Hornik, K., Stinchcombe, M., White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.

[4] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.