NLP-243: Relation Extraction from Natural Language using PyTorch

Kushagra Seth
University of California, Santa Cruz
kuseth@ucsc.edu

Abstract

The main objective of this model is to train a Multi-Layer Perceptron(MLP) to identify knowledge graph relations that are invoked in user utterances to a conversational system (in this case, according to the Freebase schema). In order to train models to predict the corresponding relations for a given speech, a training set of utterances and a set of relations have been provided.

We have trained a deep neural network model using PyTorch that outputs the related set of core-relations(labels) when given a new utterance.

1 Introduction

We can execute supervised learning on data that has labels. Basically, supervised text classification means that we have a set of instances for which we already know the labels or the ground truth. Therefore, we should supply a set of text and its labels as an input so that the computer can learn like we do. The machine then makes an attempt to learn it like a human.

Here is an *example* utterance from the data set:

Show me movies directed by Woody Allen recently.

There are two core relations that are invoked by this utterance:

movie.directed_by
movie.initial_release_date

This particular data set with which we have been provided with has 18 different classes/labels possible for each utterance.

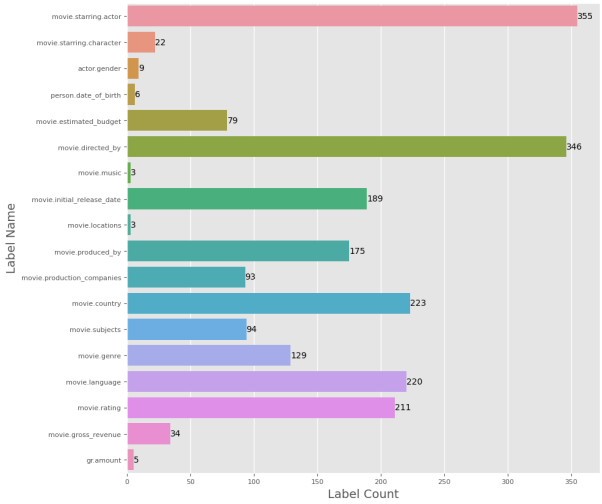

Figure 1: Horizontal Bar Plot that represents the count for each label in the data set

2 Proposed Framework

2.1 Text Pre-Processing

As is well known, machine learning requires data in the form of numbers. To basically turn text into a numeric vector, we used encoding techniques. But before encoding, the text data must first be cleaned. This process of preparing (or cleaning) text data before encoding is known as text pre processing, and it is the initial step in resolving NLP issues.

On the provided data set, we have preparing the text step-by-step.

2.1.1 Pre-Processing using Regular Expression(Reg-Ex)

Reg-Ex is a sequence of characters that defines a specific search pattern using which you can substitute patterns inside the text with the least amount of code. In simple words, Reg-Ex is used for detecting word patterns and substituting patterns inside text. The use of Reg-Ex is to pre-process the data set,

removing non-word characters, removing single words from the corpus, substituting multi-spaces by a single space in the corpus, tweet normalization, etc. As such, there is no universal list of stop words, but *spaCy* module in python has a list of stop words in 16 different languages which are removed from the data set.

2.1.2 Lemmatization

Lemmatization describes the word in such a way that its portrayal/root structure has importance. It's main use is to normalize the inflectional form of a word to its root word. So it can be used as a preprocessing step in any natural language processing application. As proposed, each word is named by a class label, that speaks to the transformation that should be applied to get the standardized form of the word. It generally alludes to doing things properly with the utilization of vocabulary and morphological analysis of words, to remove inflectional endings to return the base or dictionary form of a word, which is known as the lemma.

For Example,

If confronted with the tokens *lightly, lightless, lighters, lighting*, lemmatization would attempt to return *light* depending on whether the use of the token was as a verb or a noun.

Figure 2: Root form of a word[1]

2.1.3 Stop word Removal

Text may contain many stop words like *the, is, are, that* etc. which are common across all the documents in the corpus and carry no meaning attached to them. So, these words are filtered as we do not want these words taking up memory space and take up the processing time.

For Example,

With Stop Words	Without Stop Words
/growing-up-with-hearing-loss/	/growing-hearing-loss/
/coming-to-terms-with-hearing-loss/	/coming-terms-hearing-loss/
/the-world-of-being-hearing-impaired/	/world-being-hearing-impaired/
/echo-in-ears-from-talking-people/	/echo-ears-from-talking-people/
/listening-is-exhausting/	/listening-exhausting/
/living-with-hearing-loss/	/living-hearing-loss/
/what-is-hearing-loss/	/what-hearing-loss/

Figure 3: Text with and without Stop words

2.2 Encoding Utterances using TF-IDF

Though they are relatively simplistic, word counts are a nice place to start.

One problem with simple counts is that some words, like "the," may appear frequently and their high counts won't imply anything when they're encoded.

Calculating word frequencies is an option; by far the most well-liked approach is known as TF-IDF. This abbreviation stands for "Term Frequency - Inverse Document" Frequency, the elements of the final scores each word received.

Term Frequency gives an overview of how frequently a word or phrase appears in a document. Words that frequently appear in documents are scaled down using the inverse document frequency method. Without getting into the arithmetic, TF-IDF scores on word frequencies aim to draw attention to words that are more intriguing, such as, for instance, frequent within a document but not between documents

The TfidfVectorizer can encode new documents, tokenize existing ones, learn the vocabulary, and compute inverse document frequency weightings.

The TfidfVectorizer is used in the the mode to learn vocabulary and invert document frequencies across 3 tiny texts before encoding one of them.

2.3 Encoding Core-Relations using Multi-Label Binarizer

Predicting more than one class label is necessary in some classification tasks. This implies that belonging to a class or using a class label are not mutually exclusive. Multiple label classification, or simply multi-label classification, is the term used to describe these activities.

For each input sample in multi-label classification, zero or more labels must be produced simultaneously, and they must be needed for each output. It is assumed that the labels of the outputs are a result of the inputs.

For each sample in the data set, I have generated a vector representation of length 18, i.e., the total number of possible labels with 1 corresponding to presence of the label and 0 corresponding to label not being present.

2.4 Multi-Layer Perceptron(MLP)

The most practical variety of neural network, multilayer perceptrons are typically used to refer to the area of artificial neural networks. A single neuron model called a perceptron served as the basis for bigger neural networks.

The capacity of neural networks to learn the representation in your training data and the best way to tie it to the output variable you want to predict is what gives them their power. Neural networks acquire mapping in this way. They have been demonstrated to be a universal approximation algorithm and are capable of learning any mapping function mathematically.

The hierarchical or multi-layered structure of neural networks is what gives them their predictive power. In order to create higher-order features, the data structure can select (learn to represent) characteristics at various sizes or resolutions, such as lines, groupings of lines, or forms.

The building blocks for neural networks are artificial neurons.

These are simple computational units that have weighted input signals and produce an output signal using an activation function.

Neurons are arranged into networks of neurons.

A row of neurons is called a layer, and one network can have multiple layers. The architecture of the neurons in the network is often called the network topology.

Input or Visible Layers: The bottom layer that takes input from data set is called the visible layer because it is the exposed part of the network.

In this model the input to the MLP is the TF-IDF Matrix created using TfIdfVectorizer() as Neural Network can only process neural network data. The

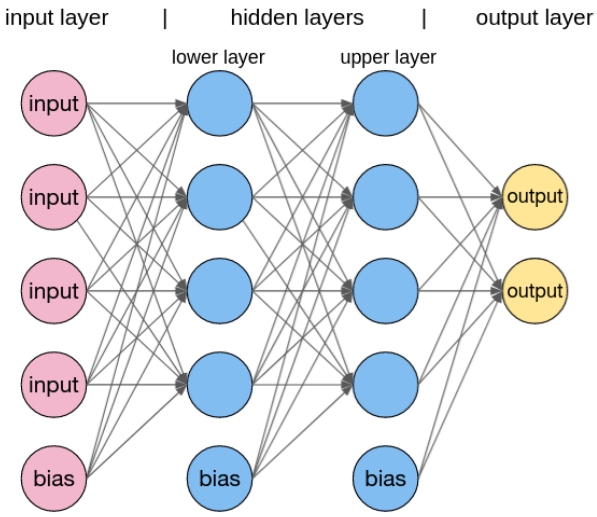

Figure 4: Multi-Layer Perceptron Model

dimension for the input layer is calculated using the size of Tf-Idf Matrix.

Hidden Layers: Layers after the input layer are called hidden layers because they are not directly exposed to the input. This layer is connected to other hidden layers or is directly connected to the output layer.

Deep learning is referred to as having many hidden layers in the neural network. They are deep because they would have been unimaginably slow to train historically but may take seconds or minutes to train using modern techniques and hardware.

In this model, we have experienced with multiple hidden layers ranging from 1 to 2.

Output Layer: The final hidden layer is called the output layer, and it is responsible for outputting a value or vector of values that correspond to the format required for the problem.

The choice of activation function in the output layer is strongly constrained by the type of problem that we are modeling.

In this problem we are doing Multi-Label Classification which is essentially a predictive modeling task that involves predicting zero or more mutually non-exclusive class labels. This is performed using Multi-Label Binarizer which returns a numpy array of labels for each class. This numpy class is then converted to a PyTorch tensor.

3 Experiments

I came up with a Baseline model with certain Hyper Parameters. Then, I have tuned the model by varying one of these parameters to increase the accuracy of this MLP based model. I got the highest accuracy with the optimizer: Adam [2] while the performance with other optimizers such as SGD and AdaGrad was way worse and could'nt even crack 0.6 F1_Score.

3.1 Experiment-1: Varying the Learning Rate

The *learning rate* or *step size* refers to how frequently the weights are updated during training.

In particular, the learning rate is a hyper-parameter that can be customized and is used to train neural networks. Its value is typically small and positive, falling between 0.0 and 1.0.

How quickly the model adapts to the challenge is determined by the learning rate. Given the smaller changes to the weights made with each update, smaller learning rates necessitate more training epochs, whereas bigger learning rates produce quick changes and necessitate fewer training epochs.

A model may converge to a less-than-ideal solution too soon if the learning rate is too high, whereas the opposite is true if the learning rate is too low.

Table 1: Fixed Hyper Parameters

Hyper Parameters	**Values**
Hidden Units	50
Epochs	20
Batch Size	16

Table 2: Varying the Learning Rate Parameter

Learning Rate	0.001	0.004	0.01	**0.04**
F1_Score	0.76	0.802	0.79	***0.814***

3.2 Experiment-2: Varying the Batch Size

Stochastic gradient descent optimization is used to train neural networks.

In this case, a prediction is made based on the model's current state, and the difference between the predicted and the actual values is used to estimate the error gradient. The model weights are then updated using this error gradient, and the procedure is repeated. The *batch size*, often known as the *batch*, is a hyper-parameter for the learning algorithm that determines how many training samples are utilized to estimate the error gradient.

Table 3: Fixed Hyper Parameters

Hyper Parameters	**Values**
Hidden Units	50
Epochs	20
Learning Rate	0.04

Table 4: Varying the Batch Size Parameter

Batch Size	**16**	32	64	128
F1_Score	***0.814***	0.791	0.784	0.802

3.3 Experiment-3: Varying the Hidden Units

The inputs feed into a layer of hidden units, which can feed into layers of hidden units, which eventually feed into the output layer. Each of the hidden units is a squashed linear function of its inputs.

Neural networks of this type can have as inputs any real numbers, and they have a real number as output. In this model, I have used one hidden layer with multiple variations of hidden units to get the maximum accuracy on the validation set.

Table 5: Fixed Hyper Parameters

Hyper Parameters	**Values**
Batch Size	16
Epochs	20
Learning Rate	0.04

Table 6: Varying the Hidden Units Parameter

Hidden Units	25	**50**	100
F1_Score	0.806	***0.814***	0.785

3.4 Experiment-4: Varying the Epochs Parameter

In machine learning, an epoch refers to one full iteration of the algorithm over the training data set. An essential hyper-parameter for the algorithm is this epoch's number. The amount of complete runs of the whole training data set during the algorithm's training or learning process is specified in terms of epochs. The internal model parameters of the data set are modified at each epoch.

Table 7: Fixed Hyper Parameters

Hyper Parameters	Values
Batch Size	16
Hidden Units	50
Learning Rate	0.04

Table 8: Varying the Epochs

Epochs	5	10	**20**	50
F1_Score	0.812	0.801	***0.814***	0.794

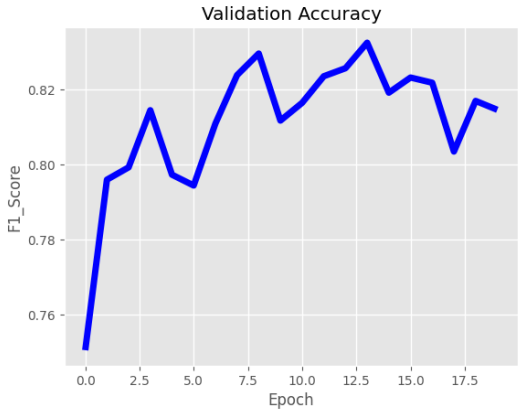

Figure 6: F1_Score vs Epoch Graph

4 Results and Conclusion

In this project, I have addressed the task of Multi-Label Classification on the data set containing Utterances and the corresponding Core-Relations. I created an MLP model using PyTorch in python for this data set and evaluated its performance by tuning different hyper-parameters.

4.1 F1_Score

Val F1_Score = *0.814*
Test F1_Score = *0.766*

Table 9: Hyper Parameters Configuration that produced highest F1_Score

Hyper Parameters	Values
Batch Size	16
Epochs	20
Hidden Units	50
Learning Rate	0.04

F1_Score is calculated using the below-mentioned formula:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Figure 5: F1_Score Formula

4.2 Average Loss

The training loss indicates how well the model is fitting the training data, while the validation loss indicates how well the model fits new data.

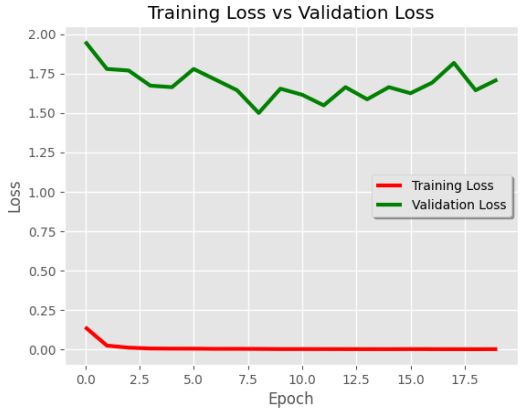

Figure 7: Visualization of Training and Validation Loss

4.3 Confusion Matrix

A Confusion matrix is used for evaluating the performance of a classification model. The matrix compares the actual target values with those predicted by the machine learning model. For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values. In this Multi-Label Classification problem we had 18 classes, so we'll have 18 matrices.

Confusion Matrices for 18 Labels

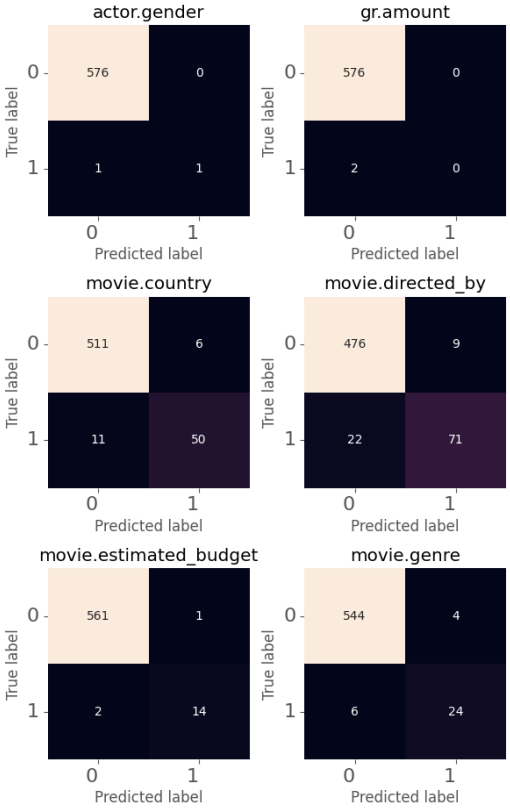

Figure 8: Confusion Matrix for Classes 0-6

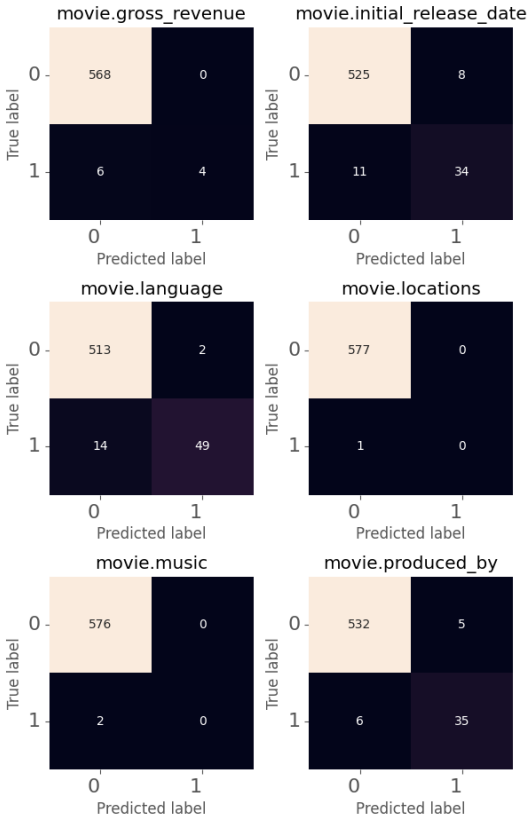

Figure 9: Confusion Matrix for Classes 6-12

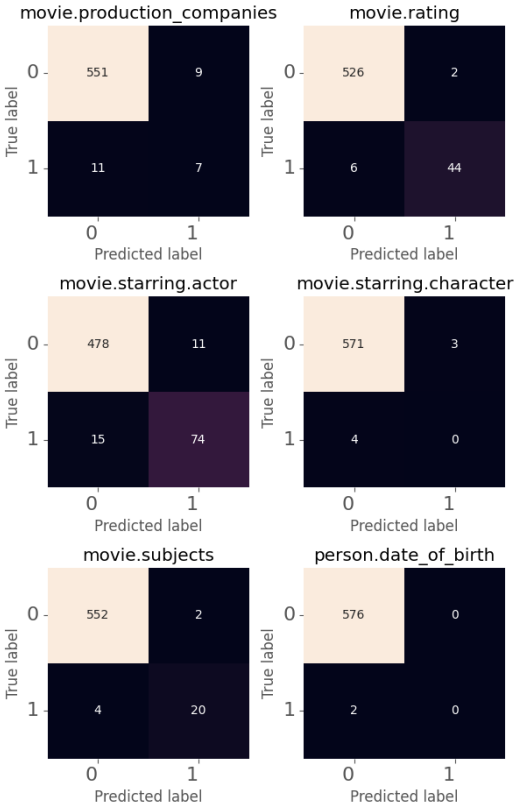

Figure 10: Confusion Matrix for Classes 12-18

Acknowledgments

I would like to thank Nilay Patel (Teaching Assistant for the this course, i.e., NLP-243) who provided me with constant support and help during this project.

References

[1]https://onlinemediamasters.com/wp-content/uploads/2015/11/Stop-Words.jpg
[2]Adam: A Method for Stochastic Optimization: http://arxiv.org/abs/1412.6980