Slot Tagging of Natural Language Utterances

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

1 Introduction

The purpose of this assignment is to train a model to identify slots in users' natural language communications with a virtual personal assistant. The users' requests are customarily fulfilled using the labeled slots and the related values. For instance, if a user inquires about movies by a particular director, we would need to know the director's name (in addition to the user's desire to learn about movies), and then we would need to send a query to the backend knowledge graph to gather data for creating a response to send to the user. 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

The ability to understand natural language is essential for conversation agents like chatbots, virtual assistants, and in-car helpers to do any task. Understanding the hierarchical semantic frame and extracting pertinent data, such as domain, intent, slots, and relations, are required for this. Large amounts of information on the web can be conveniently and abstractly organized in the form of nodes and edges thanks to knowledge graphs (KG). Many different downstream tasks, including question answering, link prediction, entity classification, and information retrieval, are being handled by the representations gained via KGs. A knowledge graph example is shown in Fig. 1 using data from the dataset's few movie examples.

For a given dataset of user utterances and slot tags, we have to create a deep learning based model and predict slot tags 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.

To form a formal problem statement,

Task Definition: *Given a labelled training dataset D with a set of utterances where words* $\{w_1, w_2, \ldots, w_n \in u_i\}$ *are labelled with* $\{l_1, l_2, \ldots, l_m \in L\}$ *, the task is to learn a prediction/classification function for an unknown utterance U that predicts a label* $l_i \in L$ *for each word in U. Here L is a set of possible slot-tag labels.*

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.

1.2 Dataset

The Dataset includes of 3291 phrases containing utterances that include information about actors, directors, producers, release dates, languages, and many movie-related terms. Dataframe structure and data entries can be seen in Fig 2.

Each word in the utterances must be assigned a slot tag in order to be classified. 28 slot tag classes altogether and 2311 and 980 samples each make up the train and test sets.

Going ahead with data exploration, we'll try to

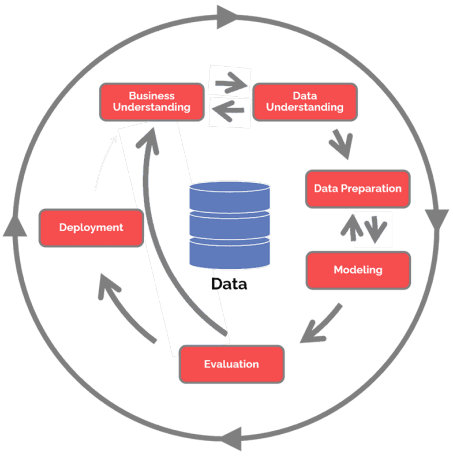

Figure 1: CRISP-DM Diagram: Inspired by Wikipedia

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.

	ID	UTTERANCES	IOB SLOT TAGS
0	569	director for pride and prejudice	O O B_movie I_movie I_movie
1	1132	list some of the best world war 2 films	O O O O O B_subject I_subject I_subject O
2	2094	please provide the name of the company that pr...	O O O O O O O O O B_movie
3	767	search for russian movies	O O B_country O
4	1199	rating g	O B_mpaa_rating

Figure 2: First five values in the training set

1.3 Visualisations

Getting a visual understanding of the dataset is essential for any data analysis. With our given data of user utterances and slot tags, 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 Fig 3, 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 Fig 4. 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.

We also observed that the dataset is extremely unbalanced, with class O having the highest frequency (10505 words), followed by I-movie (1146 words), B-movie (1016 words), and B-location(2 words),

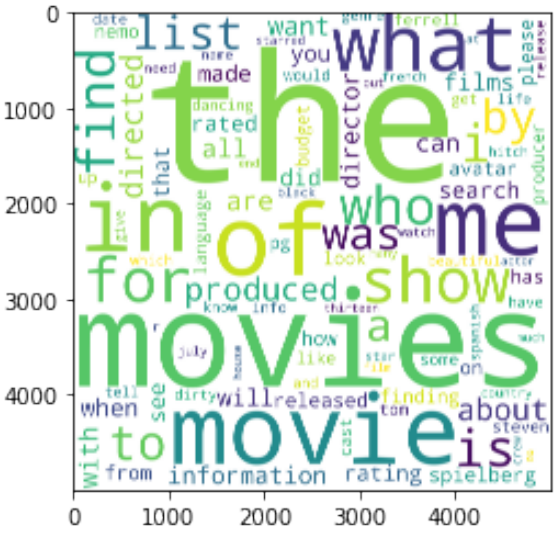

Figure 3: Word Cloud of top 100 most used words in utterances before removing stopwords

which has the lowest frequency. The frequency distribution of each class in the dataset is shown in Fig 5. Another issue is the differing word distribution between the train and test sets, as 35.7% of the test set’s words are absent from the train set.

Data imbalance was also evident with the variance in length of each utterance, ranging from 3 words per sentence to 26 words per sentence. Fig 6 gives us a graphical representation of this balance, the solution for which will be given in the following section.

2 Methodology

This section explains how to extract features from sentences and the approach used for the task of labeling sequences.

2.1 Text Preprocessing

Despite the fact that the supplied train and test data were punctuation-free, all lowercase, and cleaned, the studies employed the following processing method:

1. *Sent2Vec*(2): Some algorithms transform sentences into individual words, turning the challenge of labeling a sequence into a classification problem at the word level.

2.2 Feature Extraction

Any effective machine learning algorithm must successfully represent the incoming textual data as meaningful features. Depending on the categorization algorithm, the tests use the following features:

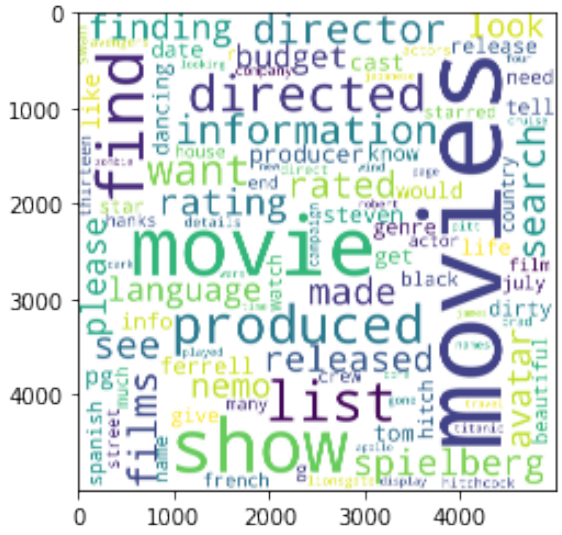

Figure 4: Word Cloud of top 100 most used words in utterances after removing stopwords

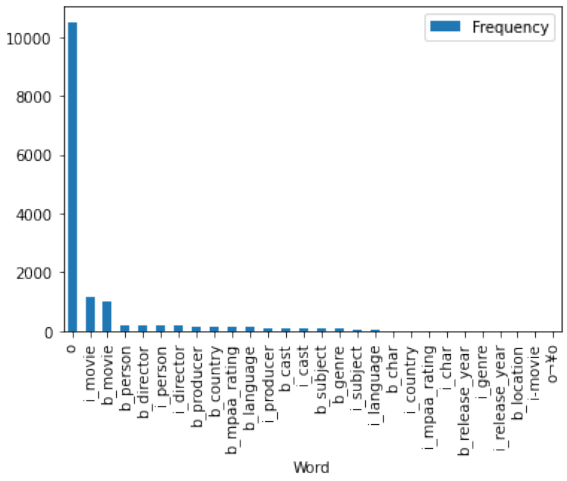

Figure 5: Word Frequency Plot

1. *One Hot Encoding*: As the name suggests, the one-hot representation begins with a zero vector and sets the appropriate element in the vector to 1 if the word is present in the sentence or document. A logical OR of the one-hot representations of the words that make up a phrase, sentence, or document results in its collapsed one-hot representation.(3)

2. *Word History (H)*: Features are manually created for word-level classification by taking a word's n-gram history into account. For experimentation, bigram and trigram features are employed. The elements of the words "Who plays Luke in Star Wars: New Hope" are seen in Fig. 5. The history is selected as the <START> token for the first word.

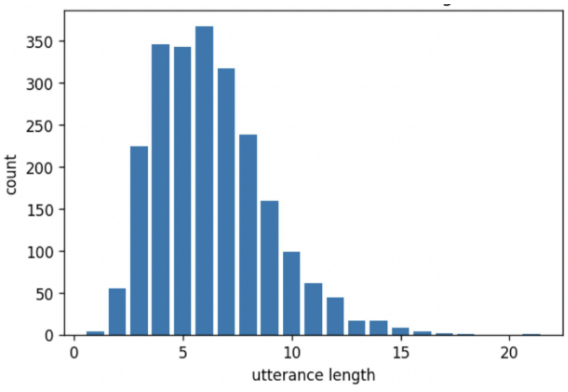

Figure 6: Distribution of number of words per utterance

3. *Word Future (F)*: I added feature information to an utterance by using future words, similar to how n-gram history was employed. The future is selected as the <STOP> token as the final word.

4. *Tag Count (C)*: If a word has a specific slot-tag in the training dataset, it is likely that it will also have that tag in an unidentified utterance. For instance, it is extremely unlikely that the word Avengers would have the slot tag B-character in the test utterance if it has the slot B-movie in the train set. The most popular tag for each term is therefore added as a feature.

2.3 Model Implementation

The basic RNNs used in language modeling read an input word and predict the next word. For slot tagging, these models are modified to take a word and possibly other features as input, and to output a slot value for each word. We considered 3 RNN based models to train and test, each of which have been described below.

1. *Baseline RNN*(4): An artificial neural network containing internal loops is known as a recurrent neural network (RNN). These internal loops cause the networks to have recursive dynamics, which introduces delayed activation interdependence among the network's processing elements (PEs). A second type of distributed representation is feasible in recurrent networks, but most neural networks use distributed representations, where information is encoded across the activation levels of numerous PEs. Information can also be represented in RNNs by changing the activation of one

or more PEs over time. These networks are also known as spatio-temporal networks since information can be encoded geographically, spanning several PEs, and temporally.

2. *Encoder-Decoder LSTM*(5):The encoder-decoder model decodes labels sequentially, one word at a time, starting with the encoder, which converts the input phrase utterance into a single context vector. During testing, the output of the preceding token is used as the input for the following word.(6)

3. *Long Short Term Memory (LSTM)*(7):Long Short Term Memory Network is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN. A recurrent neural network is also known as RNN is used for persistent memory.

3 Experiment

3.1 Train-Test Split

With an 80:20 ratio, the dataset 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(8) 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 Training

3.2.1 Baseline RNN

We tokenize the text using One Hot Encoding (OHE) and add the following tokens: <START>, <STOP>, <UNK>, <PAD> to pad the text in utterances and create uniformity in length. After tokenization and padding, we build vocab dictionary and create key value pairs with each vector. After creating an OHE matrix of 6378 dimension vector per word tensors, we create a basic 2 layered model with an embedding layer and RNN layer as our baseline. We also reshape the output tensors for desired dimensions and use Cross Entropy and Adam as our loss and optimization function respectively. Each word is then classified into one of the 28 slot classes. Tuneable hyperparameters are learning rate, batch size, epochs.

3.2.2 LSTM

As they employ a chain-like structure for processing a sequence, previous publications(9)(10) have used LSTM for problems involving sequence labeling. The LSTM uses a gating mechanism to recall important details and forget unimportant ones, which helps it maintain long-term dependence. It has three gates: an input gate, a forget gate, and an output gate. According to the batch maximum length, each utterance is padded to a predefined size for sequence labeling. The input is a batch of shape (batch-size * sequence-length), and the output is shape (batch-size * sequence-length * tag-vocab). The feature extraction technique used here is 'Word Future (F)'. The softmax over the 28 output labels represents the probability of underlying label. The tuneable hyperparameters used are batch size, learning rate, embedding dimensions, epochs. Optimiser function used is Adam. As the loss function, cross-entropy is employed.

3.2.3 Encoder-Decoder LSTM

For encoder, we have a hidden layer, embedded layer, RNN(LSTM) layer and dropout. For decoder, we have the output layer for getting the desired output dimensions that will be fed in Seq2Seq model, with a hidden layer, embedding layer, RNN(LSTM), Linear and a dropout layer. The softmax over the 28 output labels represents the probability of underlying label. The feature extraction technique used here is 'Word History (H)'. The tuneable hyperparameters used are encoder dropout, decoder dropout, batch size, learning rate, epochs, embedding dimensions, number of hidden layers (n-layer). Optimiser function used is Adam. As the loss function, cross-entropy is employed.

3.3 Hyperparameter Tuning

The range used for hyperparameter tuning are given in Table 1 for each model.

After trying all the possible permutation and combinations, we were able to identify the combination providing us with the best possible training, validation and test accuracy. Tables 2 gives us the exact figures and parameters used for each model. The model and hyperparameters that gave us the best accuracy out of all can be identified in table 3. Of course, many more iterations of training were

Model	Parameter	Range
RNN	LR	e-4-0.1
RNN	Epochs	50-500
RNN	BS	128-2048
LSTM	LR	e-4-0.1
LSTM	Epochs	5 - 50
LSTM	Emb dim	264-300
LSTM	BS	2-64
E/D	LR	e-4-0.5
E/D	Epochs	5-100
E/D	n-layers	1-3
E/D	BS	2-64
E/D	Emb Dim	264-300
E/D	Dropout	0.1-0.5

Table 1: All tuneable hyperparameters

MDL	PARAMETERS
RNN	BS:2048, LR:e-4,epochs:500
LSTM	BS:32,LR:e-2,epochs:15,Emb dim:300
E/D	BS:64,LR:e-3,E:50,ED:300,d:0.5,nL:2

Table 2: Hyperparameters giving best accuracy

run, but for the sake of the space all couldn't be included here.

3.4 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).

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

MODEL	Train Acc	Val Acc	Test Acc
RNN	0.92	0.86	0.54
LSTM	0.997	0.96	0.80
E/D	0.96	0.93	0.73

Table 3: Best results with hyperparameters in Table 2

Model	Val Acc	Test Acc
LSTM	0.96	0.80

Table 4: Model with best F1-Score

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

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

4 Results

After running multiple training and validation iterations, we came to the conclusion that the LSTM model was able to train better than the rest, with 'Word Future' feature extraction. Our observation stems from the fact that Encoder-Decoder have a one-way start to finish kind of architecture which fails to infer contextual data. It is typically presented as a linear, one way process. As for baseline RNN, due to its lack of depth and unnecessary usage of time and space, wasn't able to perform upto the mark and in time. LSTM model worked the fastest, was able to infer the contextual data and hence gave the highest F1-score, as seen in table 4. Figure 7 and Figure 8 give us a visual representation of how well the best model was performing on training and validation sets.

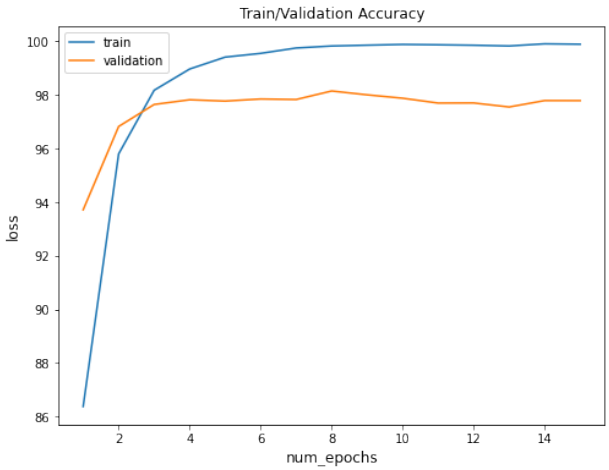

Figure 7: Training-Validation Accuracy

Given the limits of conventional machine learning models and the scarcity of annotated data, slot filling and intent detection are difficult issues. Another issue is that a biased training environment is created by a skewed class distribution. To address the sequence labeling issue, future options include the use of attention-based deep learning

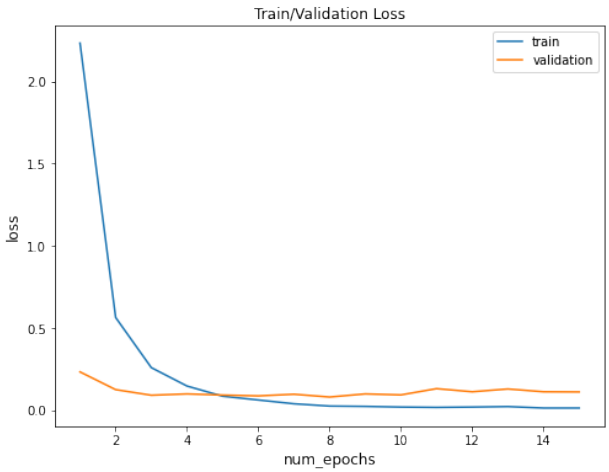

Figure 8: Training-Validation Loss

methodologies and transformer-based pretrained embeddings.

References

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

[2] M. N. Moghadasi and Y. Zhuang, "Sent2Vec: A New Sentence Embedding Representation With Sentimental Semantic," 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 4672-4680

[3] Rao, D., McMahan, B. (2019). Section 1: Introduction. In Natural language processing with pytorch: Build intelligent language applications using Deep Learning. essay, O'Reilly.

[4] McCulloch, W. S., Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

[5] Forcada, M. L., Ñeco, R. P. (1997, June). Recursive hetero-associative memories for translation. In International Work-Conference on Artificial Neural Networks (pp. 453-462). Springer, Berlin, Heidelberg.

[6] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G. (2016). LSTM-based encoder-decoder for multi-sensor anomaly detection. arXiv preprint arXiv:1607.00148.

[7] Schuster, M., Paliwal, K. K. (1997). Bidirectional recurrent neural networks. IEEE transactions on Signal Processing, 45(11), 2673-2681.

[8] 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.

[9] Gakuto Kurata, Bing Xiang, Bowen Zhou, and Mo Yu. 2016. Leveraging sentence-level information with encoder lstm for semantic slot filling.

[10] Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), pages 753–757