Predicting Hierarchical Relationship in Job Title Taxonomy

Shuang Jin
sjin1@stanford.edu

March 14, 2020

Abstract

Job title taxonomy is a digital representation of the job market knowledge. It is used in job search engines to unite and organize the global workforce. However, building a well-structured taxonomy is non-trivial. In this project, we create a classification model to predict the synonymy and hypernymy relationships between title entities to construct a complex hierarchical structure so that it helps taxonomy team to find parent and child taxa for each taxon, and place a new taxon to the proper branch.

Keywords: job taxonomy, LinkedIn, hierarchical structure

1 Introduction

In the modern era, employment-related search engine sites are an important economic driver. It has revolved the job market by connecting job seekers and talent acquirers to each other. Job seekers make online queries for openings with titles matching their professional interest, recruiters query for people with titles matching their hiring positions and the search engine sites also use user profiles to recommend jobs with titles matching their career track. This clearly shows the importance for search engines to understand job titles so to compute the best matching results in all the scenarios, and a common practice is to construct a job title taxonomy[1].

However, building a taxonomy to reflect the status quo of the job market reality is a hugely complex task that takes years of effort from a group of highly trained taxonomists. It not only requires collecting all the job title entities, but more importantly, building a hierarchical structure that provides paths that link the entities to each other.

The recent advances in Deep Learning can be used to accelerate such taxonomy tasks. Our project proposes a framework that detects the entity-to-entity relationship based on their lexical meaning and their usage among professionals and recruiters. We decode the entities' textual meanings using ELMo embeddings and also collect key metrics that observe the trend of how people use them. The model is trained on a dataset of job title entity pairs (Source Term x_{source} and Target Term x_{target})[i] and their relationship labels (y). It performs a *multiclass classification* task that predicts one of the $C = 4$ classes:

1. **Broader Term**: the target term x_{target} is a broader concept than the source term x_{source}. On the taxonomy graph, x_{target} is an ancestor of x_{source}.
2. **Narrower Term**: the target term x_{target} is a narrower concept than the source term x_{source}. On the taxonomy graph, x_{target} is a descendant of x_{source}.
3. **Preferred Term**: the target term x_{target} is the same concept as the source term x_{source}, but is deemed as a more appropriate way of expressing the concept. On the taxonomy graph, x_{target} and x_{source} reside in the same taxon (node).

[i]The input pairs are ordered. This is to say that if the source term is x_{source} and the target term is x_{target}, the label output can be different.

4. **Non-Preferred Term**: the target term x_{target} is the same concept as the source term x_{source}, but is deemed as a less appropriate way of expressing the concept. On the taxonomy graph, x_{target} and x_{source} reside in the same taxon (node).

5. **Unknown**: the target term x_{target} does not have the above relationship to x_{source}.

2 Mathematical Objective

The standard cost function for a multiclass problem with m samples and C classes of labels is as in 1, where y is the ground truth and $\hat{y}$ is the prediction:

$$J = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{C} y_j^i \cdot \log \hat{y}_j^i \tag{1}$$

However, there is a data imbalance problem that we have a lot more training data with the label *Other*, while the performance of our model depends on how well it predicts all four labels. The skewed distribution makes it more difficult for the model to learn how to classify the minority classes. Therefore we use Cost-sensitive Learning that penalizes misclassifications of the minority classes more heavily than the majority classes with a class weight λ_j for the j-th class as shown in 2:

$$J = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{C} \lambda_j \cdot y_j^i \cdot \log \hat{y}_j^i \tag{2}$$

As the class size increases, the weight decreases. The product of each class's distribution and class weight is equivalent, so our cost function is overall equally sensitive to all classes.

Class	Sample Size	Class Weight
Broader Term	100K	1
Narrower Term	100K	1
Preferred Term	20K	5
Non-Preferred Term	20K	5
Unknown	60K	2

Table 1: Data Distribution vs. Class Weight

3 Data

The taxonomy data already has a structure that indicates pair relationship. We processed the taxonomy to form a three-column data for this project: source term, target term, label2. There are 290K samples in the training set (93%), 19K in the dev set (6%) and 3K in the test set (1%). Before the modeling, the data is preprocessed in the following steps.

1. Normalization - removes trailing or double spaces and convert all words to lowercase.

2. Spellchecking - removes any data with spelling mistakes.

3. Tokenization - converts words to unique tokens (integers).

4. Padding - left pads with all zeros to convert all sequences to the same length as in Table 3.

After the preprocessing, the input layer has the shape of $(N,\ 14)$ where N is the total number of samples and 14 is the concatenated sequence length.

Source	Target	Label
Professor	Professor of mathematics	Narrower Term
Compensation specialist	Human capital	Broader Term
Information system security engineer	Information systems security engineer	Non-Preferred Term
In vitro fertilization registered nurse	Fertility nurse	Preferred Term
Account manager radiology	Sssistant researcher	Unknown

Table 2: Three-column Raw Data

Before	After
head of strategic initiatives	0 0 0 0 3 5 115 394
sales staff	0 0 0 0 0 0 0 2 57

Table 3: Tokenizing and Padding

4 Project Models

4.1 Base Model

The base model is a simple neural network with two layers. The first hidden layer is an 16-dimension embedding layer that encodes. The $(N,\ 16,\ 16)$ embedding layer is then flattened into $(N,\ 256)$. as the input of the second and final softmax layer. The model reaches 81% accuracy in validation set after 40 epochs, and then fluctuates around that accuracy level, with an increasing gap between the train and validation set.

4.2 Simple LSTM 64d and 128d Models

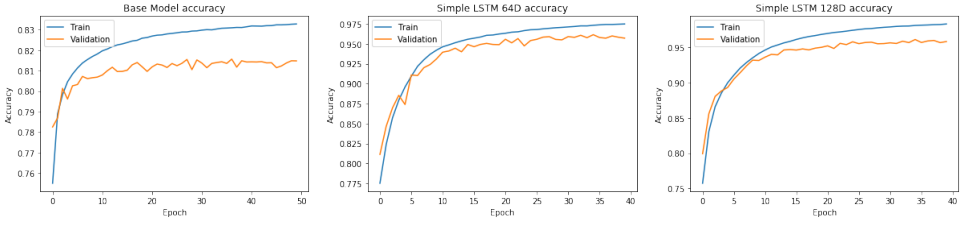
Compared to the Base Model, the Simple LSTM Models add a LSTM layer [2] of 64d and 128d respectively between the embedding and the softmax layer. The two LSTM layer dimensions are not too different in terms of their performances, but the added LSTM layer is a huge performance boost. Both reach 95% accuracy.

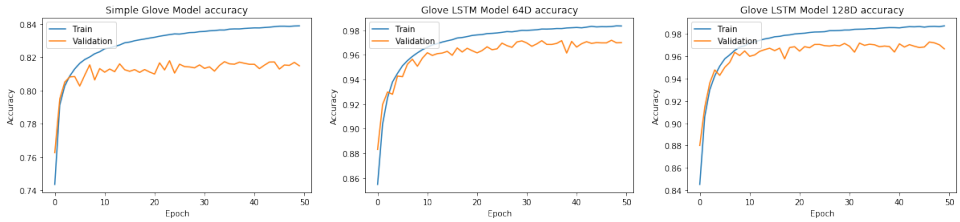
4.3 Simple Glove Model

Compared to the Base Model, the Simple Glove model replaces the self-trained embedding in the base model with the Glove 100-dimension embedding [3] (glove.6B.100d.txt). Originally, this model is anticipated to outperform the Base Model, but it turns out to have very similar performance and only has a 1% accuracy improvement.

4.4 Glove LSTM 64d and 128d Models

The Glove LSTM Models combine the Simple LSTM and Simple Glove model by applying the 100d Glove embedding followed by a 64d/128d LSTM layer. These models have the best performance across by reaching a 97% accuracy in the validation set. Similar to the Simlar LSTM Models, the 64d and 128d do not make much of a difference.





5 Observations and Analysis

5.1 Spelling errors in data

At first, the models using Glove embeddings are significantly outperformed by models with self-trained embeddings. After some investigations, I found that this is caused by spelling errors - in the raw data, about 50% of the data contains at least one spelling error. Since Glove does not recognize those spelling mistakes, the embedding layer assigned zeros to them. As job titles are usually short, these zeros have a considerate negative impact on the training. Another compound is probably the feature of the training data - the training set has 3550 unique words. Each word appears an average of 461 times in the entire set, with the top words appearing over 10K times. This indicates that the self-trained embeddings are likely to have an acceptable performance, as many words have a high repetition rate.

5.2 Model Tuning

In this project, there are two hyperparameters that have major impacts on the model performances: class weight (for loss calculation) and learning rate.

Initially, the class weights were not set as in 1. From the error analysis, we identified the trend where classes with less samples have much worse performance, so they were adjusted accordingly as shown in 2 with class weights in Table 1.

As for the learning rate, when it was set too high, the validation accuracy has large fluctuations without further improvement after a small number of epochs (very fast to learn but fails to converge). When set too low, it takes a much longer time to train. This is an expected behavior. Taking the Base Model as an example, it uses Adagrad optimization algorithm. After just a few manual experiments, I set the learning rate range to be between 0.01 and 0.1. Then I randomly sampled in this range and found out that when set at 0.03, it takes just 20 epochs to train, and converges fairly well. It is worth noting that the learning rate in this problem plays a more important role than regularization, as adding regularization term hardly reduces the variance, compared to a lowering the learning rate.

5.3 Model Comparison and Selection

Comparing all models' output, it shows that those with LSTM layer outperform those without LSTM layer by a large margin. Between the simple LSTM Models and Glove LSTM Models, Glove LSTM Models have the best performance in almost all categories and is a clear winner. The 64d and 128d Glove LSTM models are very similar with 64d being slightly more balanced across all labels. Also as the 64d model has less parameters and is faster to compute, it could have more efficiency gains in a production system. Therefore, our final selection is the **Glove LSTM 64d model**.

5.4 Observations and Future Work

Across all models, we observe a similar difficulty ranking among the five classes: $PT \approx NPT > UKN > BT \approx NT$. A possible explanation of PT and NPT being the easiest is that such relationships can often be

Model		BT	NT	NPT	PT	UKN	Precision	Recall	F1 Score
Base	BT	873	12	16	14	38	91.61%	94.38%	92.97%
	NT	6	1011	16	17	32	93.44%	93.61%	93.52%
	NPT	6	1	151	8	40	73.30%	51.01%	60.16%
	PT	3	8	6	155	33	75.61%	50.32%	60.43%
	UKN	37	48	107	114	354	53.64%	71.23%	61.19%
LSTM-64d	BT	925	0	9	7	12	97.06%	98.09%	97.57%
	NT	2	1056	2	10	12	97.60%	99.06%	98.32%
	NPT	1	0	190	14	1	92.23%	85.20%	88.58%
	PT	0	1	10	192	2	93.66%	81.36%	87.07%
	UKN	15	9	12	13	611	92.58%	95.77%	94.14%
LSTM-128d	BT	934	3	3	2	11	98.01%	97.60%	97.80%
	NT	3	1065	1	5	8	98.43%	97.98%	98.20%
	NPT	2	2	180	18	4	87.38%	89.11%	88.24%
	PT	2	3	4	193	3	94.15%	84.65%	89.15%
	UKN	16	14	14	10	606	91.82%	95.89%	93.81%
Glove	BT	871	10	15	21	36	91.40%	94.16%	92.76%
	NT	6	1008	18	18	32	93.16%	94.56%	93.85%
	NPT	7	1	144	9	45	69.90%	50.00%	58.30%
	PT	2	5	6	161	31	78.54%	48.94%	60.30%
	UKN	39	42	105	120	354	53.64%	71.08%	61.14%
Glove-LSTM-64d	BT	938	1	5	5	4	98.43%	99.05%	98.74%
	NT	0	1069	1	5	7	98.80%	98.71%	98.75%
	NPT	1	3	188	14	0	91.26%	92.61%	91.93%
	PT	0	2	6	195	2	95.12%	85.15%	89.86%
	UKN	8	8	3	10	631	95.61%	97.98%	96.78%
Glove-LSTM-128d	BT	939	0	4	4	6	98.53%	98.02%	98.27%
	NT	1	1069	0	3	9	98.80%	98.62%	98.71%
	NPT	0	1	190	11	4	92.23%	93.14%	92.68%
	PT	1	2	7	192	3	93.66%	86.49%	89.93%
	UKN	17	12	3	12	616	93.33%	96.55%	94.92%

BT = Borader Term
NT = Narrower Term
NPT = Non-Preferred Term
PT = Preferred Term
UKN = Unknown
Highest among all models

Table 4: Model Comparisons

decided from the language features alone. Using well-trained embeddings, machine can achieve this task as well.

For humans, UKN is often the easiest class among the five, but the models still find it difficult. This could be caused by the fact that humans are able to utilize a variety of knowledge such as the associated industries and skills of each class. Therefore, a possible extension is to mine the related entities such as industries and skills to improve the performance of UKN.

As for PT and NPT, humans also tend to have more troubles making the decision. A common practice in the human taxonomy team is to refer to the popularity of the terms. For example, if more people use Term A than Term B in their resumes, it is a strong indicator that A is more preferred. Such features are missing in this project but should be added in the future.

Another observation is that the labeled data is "sector-biased" - it has a lot more data in the "white-collar" industries. It is worth doing additional analyses to uncover model performance differences in for each sector and rectify the biases accordingly.

References

[1] Mamadou Diaby and Emmanuel Viennet. Taxonomy-based job recommender systems on facebook and linkedin profiles. *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–6, 2014.

[2] Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.