CriticalRiver Technologies

Deep Learning and Text Classification in Tickets

Internship Use Case Document

Pradyun Magal

Contents

1. **Introduction** **3**
   - 1.1. *Problem statement* *3*
   - 1.2. *Objectives and Scope* *3*
   - 1.3. *Background on Reinforcement Learning and Language Models** *3*
2. **Literature Review** **4**
   - 2.1. *Key Concepts of Reinforcement Learning and Language Models** *4*
   - 2.2. *Review of Relevant Research and Studies* *4*
3. **Data Collection and Preprocessing** **5**
   - 3.1. *Description of the Dataset Used* *5*
   - 3.2. *Data Preprocessing Steps, if applicable* *5*
4. **Model Selection** **6**
   - 4.1. *Explanation of the Chosen Model* *6*
   - 4.2. *Training the Model, if applicable* *6*
5. **Reinforcement Learning with Human Feedback*** **7**
   - 5.1. *Overview of the Approach for RL with Human Feedback* *7*
   - 5.2. *Collection and Incorporation of Human Feedback into the Language Model* *7*
6. **Application Development** **8**
   - 6.1. *User Interface (UI) Design and Functionalities* *8*
   - 6.2. *Integration of Language Model with the Application* *8*
7. **Results and Evaluation** **9**
   - 7.1. *Presentation of Application Results* *9*
   - 7.2. *Evaluation of Application Performance and User Feedback* *9*
8. **Discussion and Conclusion** **10**
   - 8.1. *Key Findings and Insights* *10*
   - 8.2. *Limitations and Challenges Faced* *10*
   - 8.3. *Conclusion and Future Directions* *10*
9. **References** **11**

Table of Figures

1. Introduction

1.1. Problem statement

We are presented with a typical Text classification problem of categorizing pieces of text based off what they say. As someone who has worked in IT Support ticketing, organizing or routing tickets to the appropriate team can be a tedious and repetitive process, especially when the client needs to get re routed multiple times. Having a person need to manually guide a ticket is an additional task that can frankly be avoided in today's day and age of technology. We felt the best way to handle this is to train a model that can automatically route IT support tickets to their appropriate group based on the problem description.

1.2. Objectives and Scope

Our objective in this case is to make an accurate and efficient model that takes in Ticket descriptions as an input and gives a predicted support group as an output. The scope of this project is intra-organization, I say this because ticketing needs and groups vary from company to company or service to service and it is almost impossible to train a model that works universally with every ticketing system.

1.3. Background on Reinforcement Learning and Language Models*

Reinforcement learning involves providing a computer with the correct set of corresponding labels to their input features, and having it learn the mathematical relationship between the feature and label set through repetition. Language models aim to predict relationships between words in a given context, be it generate prediction of words that could come after or classify pieces of text.

2. Literature Review

2.1. Key Concepts of Reinforcement Learning and Language Models*

- Word to Vector
- Bag of Words
- Reward Functions
- Value Functions
- Trial and Error Search Delayed Reward
- Tokenizers
- BERT and other Encoders

2.2. Review of Relevant Research and Studies

Over the prior weeks I spent a lot of time doing research on how computers process and handle text or human languages. I read more into topics such as NLP, Deep learning and different types of unsupervised clustering and association techniques. I read into how when computers are faced with text, they often need to derive some meaning out of it, and unlike people they don't have a complex understanding of meanings and contexts. As a result methods such as converting words to Euclidean vectors and corresponding the word's "similarity" with their distance apart as vectors. I also read into Deep learning networks and how Layers of Neurons can emulate how human brains learn in machines.

3. Data Collection and Preprocessing

3.1. Description of the Dataset Used

The dataset was a very large and thorough one but only two columns,Group and Description, would be used(Group being the Label and Description being the label). It was a CSV file filled with real life data and tickets from an organization. This means that it would have all of the typical things most data scientists consider “imperfect” or “unwanted” in the dataset such as Names, Emails, typos, numbers and special characters. There was also a very large imbalance in the set, with one of the labels taking up more than all of the other labels combined.

3.2. Data Preprocessing Steps, if applicable

This step arguable was the most difficult in this project. Our first step was to handle the actual text content. At first I tried using regex to clean out some of the special characters but the Tensorflow BERT pre processor seemed to take care of this for me. Removing special characters and numbers (eg. “@#$%^”) seemed to actually decrease accuracy, as certain things like emails and usernames were now skewed or somehow mushed together. I tried doing a lot of testing around with what works best in the data cleaning, between making all text into their python.lower() form and trying to leave in email addresses, we found out the best strategy was to:

1. Remove all NER(Name or Entity Recognizers)
2. Make All text lowercase
3. Let the BERT Preprocessor from Tensorflow Hub do the rest

This seemed to work much better as our testing accuracy went up dramatically. It also removed any possibility of dataset bias while letting a preprocessor (better than any I could make) handle the needed changes to the text. While the following steps did help with making sure our raw string data was in a better place to feed the model, it did not account for a very large issue with our dataset; Imbalance. The dataset we were looking at was simple in terms of how it was laid out, there was the descriptions which served as our feature column and the Assigned group(s) which was our Label column. The issue lied in the amount of Ticket descriptions under the label “I-Prompt”. This label had 5537 Corresponding features, while some labels only appeared as little as little as 200 times. After thinking about it and weighing many options ranging from SMOTE to simply deleting a large portion of the imbalanced/excess data, we felt that since text and natural language data Is hard to fix imbalances with in terms of filling or deleting entries it was best to create two models. The main and larger model would have any label with more than 1000 appearances, the smaller model would have any label with less than 1000 appearances, and any value with less than 100 appearances would be dropped completely. The idea was that we could simply place any predictions from the larger model that it couldn’t resolve into the smaller model, and from there find the right label.This approach gave us the best yield in terms of accuracy and making sure our model wouldn’t predict the top two labels each time.

4. Model Selection

4.1. Explanation of the Chosen Model

The model we decided to proceed with was a standard Neural Network trained by us via Keras and Tensorflow. We made two different Neural Networks, one for the large model another for the smaller model. Both models had 8 activation neurons in each layer, as the Large model needed 7 for its own labels, one for predictions to be sent into the smaller, and the smaller model had 8 different classes to predict. Both models had an array of length 8 as their output, the index with the highest value would determine the final predicted value. I also added a dropout layer in order to predict overfitting, the dataset is rather large and we planned to run many epochs as language models need a lot of examples to understand text.

4.2. Training the Model, if applicable

Training the model was straightforward, we simply used sklearn's train and test split library to split our data into appropriately sized chunks and utilized the stratify feature to ensure that we were training and testing our data in a balanced manner. I ran both models through their training sets for about 30 Epochs each, the accuracy on the larger model hit around 75% accuracy and the smaller 78%. While these may not be as accurate as we wanted, these were significant improvements from prior attempts with different data cleaning methods and even more epochs. Splitting the project into two models drastically improved accuracy, as before the model would essentially almost always predict the two most appearing labels.

5. Reinforcement Learning with Human Feedback*

5.1. Overview of the Approach for RL with Human Feedback

NA

5.2. Collection and Incorporation of Human Feedback into the Language Model

NA

6. Application Development

6.1. User Interface (UI) Design and Functionalities

The user interface is a simple Google Colab notebook where the user can input their ticket description and it will run it through both models and print the predicted label.

6.2. Integration of Language Model with the Application

Simply saved the model to Drive, and instantiated it to the notebook. The models are ran each time the user sends a query.

7. Results and Evaluation

7.1. Presentation of Application Results

The application runs very well, I tested around with many different existing tickets and test ones the model has not seen before and it gave great predictions each time.

7.2. Evaluation of Application Performance and User Feedback

TBD during presentation.

8. Discussion and Conclusion

8.1. Key Findings and Insights

The biggest findings for me was how much data prep is needed when training models from scratch, any small errors can completely ruin your progress. I also learnt the amount of computing power needed to train these models, I ended up having to use GPUs on Google Colab for them to train fast enough to complete the project on time. But the most valuable lesson I learnt was the value of having balanced data. Data imbalances as a topic is something I had minimal experience with in my ML classes, but this project really brought to my attention the impact it has on Models, especially language models.

8.2. Limitations and Challenges Faced

The biggest drawback was needing to occasionally purchase compute units from Google for Colab, asides from that there weren't any major issues with the project.

8.3. Conclusion and Future Directions

Other than the paywalls there weren't any notable hurdles with this project.

9. References

9.1. List of All the Sources and References Used

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

https://aws.amazon.com/what-is/neural-network

https://www.javatpoint.com/unsupervised-machine-learning

https://youtu.be/lhufOy2W3Ps

https://aws.amazon.com/what-is/deep-learning/

https://www.ibm.com/topics/deep-learning