HW 1: Relation Extraction

Kiara LaRocca | klarocca@ucsc.edu

Abstract

Deep learning is an important part text classification, as deep learning has become incredibly accurate and effective. This assignment explores the performance of a Multilayer Perceptron (MLP) neural network for multi-class classification. The model reached 79 percent accuracy at its best performance. Future research consists of better documenting hyperparameters to further enhance performance.

1 Introduction

The problem I am trying to solve in this homework assignment is a supervised learning multi-class classification problem. The task is to correctly classify core relation classes given an utterance from a specific film schema, detailed below. I will use PyTorch to train and tune a deep neural network model and output the associated set of relations. Some techniques I will try include:

- Multilayer Perceptron (MLP)
- Different core features (bag of words, word embeddings)
- Different loss functions (ReLu, Leaky ReLu)
- Preventing overfitting (via dropout and weight-decay regularization)

1.1 Describing the Dataset

As mentioned above, the dataset used for this task was generated based on the film schema from Freebase knowledge graph. Freebase is a collaborative knowledge base created by the community that captures relationships between real-world entities. The film schema includes information about films, like movies, directors, actors, producers, and genres, and the corresponding relationships between them.

There are three CSV files included: hw1-train.csv, hw1-test.csv, and exampleSubmission.csv. Each of these files contains an 'ID,' and either 'Utterances' or 'Core Relations.' The training file contains all of these.

1. **hw1-train.csv** contains 'Utterances' and 'Core Relations,' which are the inputs and outputs of the dataset, respectively. Utterances are text representations of user inputs or dialogue relating to films, and are of varying lengths. Core relations are the target labels that represent the relationship or category of each utterance. Some utterances have multiple core relations per instance. There are 2312 instances and 47 classes in the training dataset. The data has also not yet been processed for things like separating classes or tokenizing utterances.

	A	B	C
1	ID	UTTERANCES	CORE RELATIONS
2	0	who plays luke on star wars new hope	movie.starring.actor movie.starring.character
3	1	show credits for the godfather	movie.starring.actor
4	2	who was the main actor in the exorcist	movie.starring.actor
5	3	find the female actress from the movie she's the man	movie.starring.actor actor.gender
6	4	who played dory on finding nemo	movie.starring.actor movie.starring.character
7	5	who was the female lead in resident evil	movie.starring.actor actor.gender
8	6	who played guido in life is beautiful	movie.starring.actor movie.starring.character
9	7	who was the co-star in shoot to kill	movie.starring.actor
10	8	find the guy who plays charlie on charlie's angels	movie.starring.actor movie.starring.character
11	9	cast and crew of movie the campaign	movie.starring.actor
12	10	cast and crew of the campaign	movie.starring.actor

Figure 1: hw1-train.csv

2. **hw1-test.csv** contains only 'Utterances,' and is the file that 'Core Relations' will be predicted for. This file has also not been processed yet, and contains 943 instances. The number of classes depends on the predictions of the model.

	A	B
1	ID	UTTERANCES
2	0	star of thor
3	1	who is in the movie the campaign
4	2	list the cast of the movie the campaign
5	3	who was in twilight
6	4	who is in vulguria
7	5	actor from lost
8	6	who played in the movie rocky
9	7	who played in the movie captain america
10	8	cast and crew for in july
11	9	who is in movie in july
12	10	who's in star wars episode four

Figure 2: hw1-test.csv

3. **exampleSubmission.csv** is the last CSV file included, and it contains the data that we are trying to predict - the 'Core Relations' column. This file is only meant as a guide for our own submission, which will come later. The size of this particular file is not important, but the size of the actual *submission.csv* file in the end will be.

	A	B
1	ID	Core Relations
2	0	movie.initial_release_date
3	1	movie.initial_release_date
4	2	movie.initial_release_date
5	3	movie.initial_release_date
6	4	movie.initial_release_date
7	5	none
8	6	movie.initial_release_date movie.starring.actor
9	7	movie.initial_release_date movie.starring.actor
10	8	none
11	9	movie.initial_release_date movie.starring.actor
12	10	movie.initial_release_date movie.starring.actor

Figure 3: exampleSubmission.csv

2 Models

The main embedding methods that I used were **CountVectorizer** and **GloVe**. I picked these two embedding methods because I wanted a simple method as I am new to PyTorch, but I wanted to challenge myself to try something new. I mainly experimented with CountVectorizer for this project, but I did play around with GloVe for the experience of trying it.

CountVectorizer is based on the **Bag of Words (BoW)** model, and is easy to understand, but there is no semantic information captured. The input text *(Utterances)* is tokenized into individual tokens (words) and a vocabulary is made of all the unique tokens. The tokens are converted into vectors, whose size is equal to the frequency of the token.

GloVe is a method that uses pre-trained word embeddings, and is able to capture semantic meaning/relationships. GloVe calculates how often *word-pairs* occur in a context window (a number of words around the word-pair), and creates vectors by factoring in that co-occurence. GloVe generates dense word vectors that usually have about 300 dimensions. If you introduce *new* words, they will not have embeddings unless you manually handle them.

2.0.1 CountVectorizer

This is a simple neural network with three layers: an input layer, hidden layers, and an output layer. The activation function is **Leaky ReLu**, and the loss function is **CrossEntropyLoss**. Dropout is used to prevent overfitting.
The training loop uses mini-batch gradient descent with an epoch loop, a batch loop, and progress logging. Tuneable hyperparameters include learning rate, hidden size, batch size, number of epochs, dropout, weight-decay, etc.

3 Experiments

The experiments that I did were not originally well documented, as I did not realize how important changing the smallest little thing would be. I should have immediately approached this assignment with the mindset of detailing every change that I made, but these experiments are missing some information.

To begin with, the data-set split that I used was an 80-20 split. I began by splitting the data into features (x) and labels (y) and then vectorizing the features and encoding the labels. This step basically converts both x and y into numerical formats, where the utterances have a numerical vector space in a matrix, and the labels have a number corresponding to each unique class. Splitting the data allows the model to train on 80 percent of the total training data, where the remaining 20 percent will be used to test the data before it is evaluated on the test set. Setting a seed makes the data split reproducible, and changing the split could affect the accuracy.

The method for selecting hyper-parameters was incredibly trial-and-error at first. I have only ever tuned hyperparameters with guidelines, so deciding what to tune myself was an interesting experience. In the future, I would evaluate the output of the model more before changing hyperparameters, but in the beginning, I just changed numbers without thinking or documenting most of the changes just to see what would happen. The very first model that I sumbitted was one of my best, and the hyper-parameters were default parameters. I also have never dealt with large amounts of data imbalance, so I tried GloVe and K-fold Cross Variation to deal with this, but only at the end of my initial experimentation.

When my accuracy was stagnant, I tried to implement **Bayesian Optimization** to find the best hyperparameters for the model. I ended up doing too much, and subsequently got "lost in the sauce." There were more options for each tuneable parameter than there arguments to pass through, and I felt that I was not making enough progress. In future experiments, I know I must start small and only change one or two things at a time. The package I used for this was **Optuna**, and while the code is not in the *run.py* file, it will be included in my submission.

I also think that if I had more time to learn about ways of evaluating the metrics of a model, I would have tried more. As such, I was still learning, and I was mostly trusting that everything seemed okay. I uploaded my submissions to Kaggle without really running any tests to evaulate to see which hyper-parameters may have been limiting the accuracy. One of the ways I did, however, was to visualize the count frequency of core relations after the model was evaluated on the test dataset.

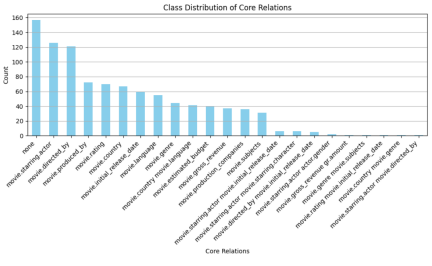

Figure 4: Class Distribution

I randomly chose a lot of the test set submissions not knowing how good the model was, and I know that I can do that differently now. Things I *should have* checked for include precision and recall, by using the F1-Score. Although we have read about the F1-Score and talked about it in classes, when I was evaluating my model, I was still out of my depths with knowing what I could use, and I ended up not using this. I added F1 score for the last iteration of my model before total project submission, and I will talk in more detail about this below:

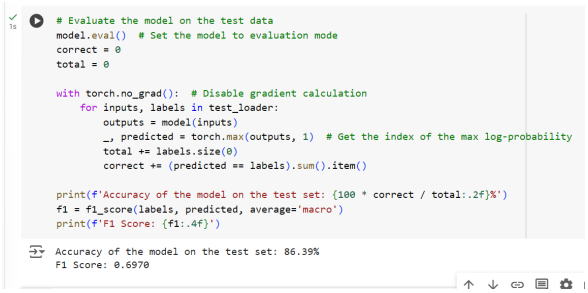

Figure 5: F1 Score

I also tried to experiment with K-fold Cross Variation, Stratified K-fold Cross Variation, and with how many hyper-parameters I was using at a time. I will continue the **Experiment** section below, as I discuss results.

4 Results

In terms of accuracy, the best accuracy I had when uploading was with the first model I made. The model performed well on train, validation, and test sets. It was not great on the final test set, but overall, for a first model, it did pretty well. It reached 87 percent on the performance for the training set, and stayed around that same percentage. Before I included *dropout*, the model did hit about 97 percent on accuracy for the test set, but was overfitting, and had an average performance on the unseen test set. The first thing that I did was implement drop out and weight-decay L1 regularization. I did not know I needed to add both, but I did not want my model to overfit, and so I added default values to both. I also changed some of the values, like the numbers for hidden size and batch size, and the numbers went down. While I was searching for ways to keep track of how the model was performing, I came across Bayesian Optimization. Using this, my model never really improved as much as I wanted, but I got to see how the results changed using different hyper-parameters.

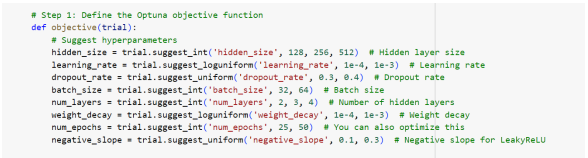

Figure 6: Optuna Objective Function

The results gave me a list of all of the hyper-parameters I was tuning and listed the best ones.

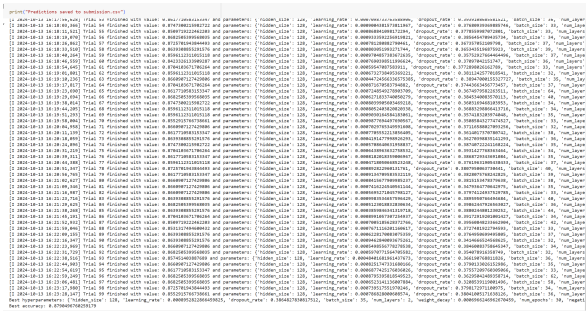

Figure 7: Optuna Results

My problem with this is that obviously I did too much (it does not even fit in the screenshot) and I was mostly doing it for entertainment rather than having a specific goal in mind. This tool is a great one to use, if you have a good reason to use it. I did not know how to optimize my use of it at first, and ended up going back to trial-and-error. Each parameter only took so many arguments, and I had more options for each than I could have, so the results were super unreliable for me. I would approach this differently in the future.

Let's return back to the F1 Score mentioned above. Like before, I should have used this metric well before the deadline, but I was still exploring the different ways to even tune hyper-parameters, let alone evaluate how they are doing. So the F1 score is the harmonic mean of precision and recall, and that depending on the application, could be acceptable or not. 0.6 is not an awful F1 score, as it does suggest balance between precision and recall, but there is room for improvement. Using this score earlier could have given me a better idea of how well the model was truly performing.

One of the ways I tried to visualize the results, again close to the deadline but this should have been evaluated earlier as well, is a confusion matrix:

This confusion matrix is one of the strangest I have seen, but it shows that the classification model is performing fairly well. There is a lot of blank space in this matrix, because some labels are multi-class. Something that might have improved performance would be splitting that up, because if the model predicts class 1, but not class 2, it is incorrect even though it is halfway there. There are also a lot of 'None' values in the prediction frequency distribution. However, despite all of this, confusion matrices are a good indication of true positives, true negatives, false positives, and false negatives. A strong diagonal line, visible in Figure 8, shows that the model is accurately predicting classes with a high degree of precision and recall. It could be higher, but if there was no line at all, I would know that the model is not predicting anything well. The trend shows that the model is doing okay, and that perhaps problems could improve with further tokenization or tuning of hyper-parameters.

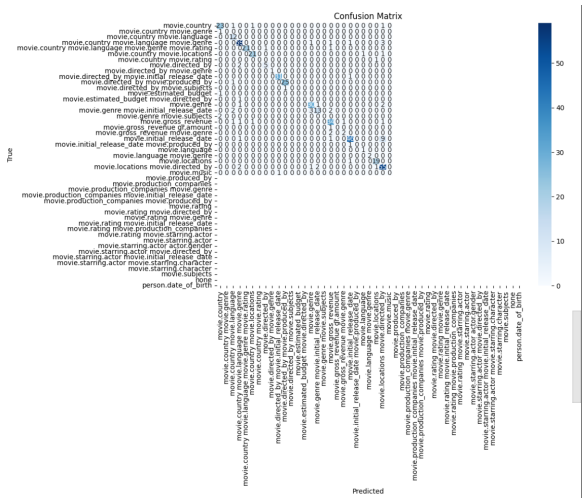

Figure 8: Confusion Matrix

The reason that most metrics were not documented was because they either did not change at all, or very slightly, or they did so poorly I was not inclined to follow through on fixing them. When I tried to use GloVe, the accuracy I got on my training and test set was about 13 percent. Now, obviously, there is fine-tuning to do there. There is also the chance that something got really messed up. Since I had a working model, it was mostly for me to just try using it, instead of including it in the experiment. I wanted to use it for the reasons listed in the Introduction/Models sections, but it was not really part of the homework assignment.

Overall, I believe my model really only performed well because I did not have a great idea of what I was doing, and in doing so, was causing more harm than good. In the future, for this section, I would have a better idea of what made each model good, and I could elaborate on why. Since I just have the one working model, I think it worked well because I did not over-complicate it. It was a solid model that performed averagely. My results were all over the place because *I* was was all over the place doing this assignment. In the future, I will have a better plan, and conduct better experiments. At one point, I was 15th on the leaderboard, and then I dropped to 20th, and was not able to improve my model because I had so many different versions and half-finished models.

5 References

I cannot get the bibliography to work in LaTex so here is the manual bibliography:

1. Glove: Global Vectors for Word Representaion
   Pennington, Socher, Manning
   October 14, 2024

2. CountVectorizer
   scikit-learn
   October 14, 2024