CSE244 Homework 1 Report

Molly Can Zhang, Feb 11 2020

Introduction **2**
1.1 Data and Label Description 2
1.2 Basic Data Stats 2

Preprocessing **4**
2.1 Train, validation and test split 4
2.2 Process Labels 4
2.3 Create vocabulary for ngram model 4
2.4 Create vocabulary for fasttext word embedding 5
2.5 Bag of Character and Character Embedding 5
2.6 Mining Movie Names and Person Names 5

Model Descriptions **5**
3.1 Loss Function 6
3.2 BaseModel 6
3.3 BaseModelNGram 6
3.4 GRU 6
3.5 MultiLayerPerceptron 7
3.6 BertMovie 7
3.7 Ensemble Method - Naive Ensemble 7
3.8 Ensemble Method - Multitask Ensemble 7

Training Setup **8**

Performance Comparison **9**
5.1 Baseline 9
5.2 Ngram results 9
5.3 Word embedding results 9
5.4 Ensemble results 10
5.5 Phrase representation results 10
5.6 Character level embedding results 11
5.7 Bert Embedding. 11

Links and References **12**

Introduction

1.1 Data and Label Description

	raw_text	raw_label
1939	movies by tom hanks	movie.directed_by
1670	what is the ratings of action movies	movie.rating movie.genre
2801	find how much money shark tales made	movie.gross_revenue
2797	can you please tell me about the revenue for t...	movie.gross_revenue
654	pretty girl movie genre	movie.genre
168	the godfather's budget	movie.estimated_budget

The table above shows 6 examples of the data. The feature of the data is a sentence, or utterance, and the label describes the relations present in the sentence. When there are no relations present, the label is “NO_REL”. There could be multiple relations present for one sentence, for example, “what is the ratings of action movies” has two labels, “moving.rating” and “movie.genre”.

1.2 Basic Data Stats

In the training data, there are 3338 utterances, and in the test data, there are 1000 utterances. In the 3338 training data, there are 46 unique labels. The most common and least common labels are as following:

Most common labels:

```
movie.directed_by                374
NO_REL                           333
movie.starring.actor             322
movie.rating                     203
movie.initial_release_date       192
```

Least common labels

movie.country movie.subjects	1
movie.award_nom movie.starring.actor award.ceremony	1
movie.media movie.language movie.country	1
movie.rating movie.directed_by	1
movie.production_companies movie.star_rating	1

Preprocessing

2.1 Train, validation and test split

I split the training data into 80% train (2635 samples), 10% validation (333 samples) and 10% holdout test (333 samples) randomly. I have both validation and holdout test because I don't want to overfit to validation or to public leaderboard, so I reserved the 10% holdout test data only to be used at the very last step when I am deciding on which model to use for final submission.

2.2 Process Labels

I converted each label text into numerical presentation in this way:

1. I split the label into multiple labels by using empty space as delimiter.
2. I gather all possible labels and sort them alphabetically. Now I have an ordered list of all 46 labels.
3. For each sample, I create a many-hot vector of length 46 to represent its label. In the label vector, at each label index, it's a 0 if this sample does not have this label, and it's a 1 if a sample does have this label.

Finally, the text labels are converted to many-hot vectors like shown below:

	raw_label	label
0	movie.starring.actor movie.starring.character	00000000000000000000000000000011000000000000000
1	movie.starring.actor	00000000000000000000000000000010000000000000000
2	movie.starring.actor	00000000000000000000000000000010000000000000000
3	movie.starring.actor actor.gender	01000000000000000000000000000010000000000000000
4	movie.starring.actor actor.gender	01000000000000000000000000000010000000000000000

2.3 Create vocabulary for ngram model

I created 1-gram (bag-of-words), bi-gram and tri-gram vocabulary for the dataset by doing the following:

For 1-gram, I split the utterance text into individual words by using either space " " or single quote " ' "as delimiter. The rationale of using space is obvious, and the reason for using single quote as a delimiter is to split words like "I'll" into "i" and "ll", is because this performs better in

subsequent experiments. I then collected the resulting 1977 vocabulary (unique words after splitting) and saved it to a list for later.
For bi-gram, I turn an utterance such as “show all sy fi movies for twenty twelve” into bigram of “show_all”, “all_sy”, “sy_fi”, “fi_movies”, “movies for”, “for twenty” and “twenty twelve”. I obtained a vocabulary of 6995 bigrams.
Likewise, for tri-gram, the utterance “show all sy fi movies for twenty twelve” is turned into tri-gram representation of “show_all_sys”, “all_sy_fi”, “sy_fi_movies”, “fi_movies_for”, “movies for_twenty”, “for twenty_twelve”. I obtained a vocabulary of 10113 for tri-grams.

2.4 Create vocabulary for fasttext word embedding

I downloaded pretrained fasttext word embedding from fasttext.cc, more specifically I downloaded “crawl-300d-2M-subword.zip”, a 2 million word vectors trained with subword information on Common Crawl (600B tokens).
Then I filtered the 2-million word vocabulary from fasttext down to the 1977 1-gram word vocabulary from this dataset. 1878 of the 1977 words are found in fasttext and the rest of the words not found in fasttext vocabulary, and these words embedding are randomly created following a normal distribution.

2.5 Bag of Character and Character Embedding

I created an additional data representation by treating each character in an utterance as an independent unit. This results in a much smaller vocabulary of size 45. I created data presented as “bag of character” as well as “character embedding” similar to bag of words and word embedding.

2.6 Mining Movie Names and Person Names

I downloaded an additional dataset from kaggle called “The movies dataset”, from which I extracted names of all known movies, actors and actresses, producers and film companies. Then from our homework dataset, I found utterances that contain movie/person names, and then replace those names with a phrase representation. For example, the utterance “star of modern family” is now converted to “star of modern_family”, representing modern_family as one word instead of two. By doing this, I am hoping to obtain a more concise representation of the utterance and to prevent the models from interpreting the movie and person names by the individual words. This improved the performance of the ngram model as will be shown later.

Model Descriptions

All models used in this homework are in model_utils.py.

3.1 Loss Function

All models (except for multi-task ensemble) share the same loss function, BCEWithLogitsLoss, which is a combination of sigmoid and binary cross entropy loss. In all models, the last layer is 46 neurons, each representing a label. The sigmoid of each neuron's output represents the probability of that underlying label being true in the sample. The loss is to make sure that when a label is present (has a value of 1), then the neuron's output (logits) should be as high as possible such that the sigmoid of the output is close to 1, and likewise, when a label is not present (has a value of 0), the neuron's output logits should be low such that the sigmoid of output is close to 0.

3.2 BaseModel

This model takes word embedding as input, the input dimension is the same as word embedding dimension, which is 300 in my model. It outputs the 46 label logits directly. It's essentially a logistic regression model.

3.3 BaseModelNGram

This model takes ngram (a.k.a bag of words) as input, one can specify whether to use 1gram, bi-gram or trigram by passing the "ngram" parameter, which by default is 1, meaning to use 1-gram. The input dimension is the same as ngram vocabulary, it has a hidden middle layer of neurons with ReLU activation function, which is followed by the 46 output neurons. Batch normalization and/or dropout are also implemented in the hidden layer. Batch normalization is shown to improve the convergence time, and also serves as regularization. Therefore I used batch normalization.

3.4 GRU

I experimented with both LSTM and GRU, and it turns out that these two recurrent models have similar performance. Since GRU is simpler and trains faster, I chose to use GRU at all places. Just like the base model, GRU model takes in word embedding as input, and pass word embedding word-by-word into a GRU and the hidden units after the last word are used to predict the final output.

3.5 MultiLayerPerceptron

Instead of GRU, I also used fully-connected neural network, or “multi-layer perceptron”. In this model, sum of individual word embeddings is being used as input, and then the sum is fed forward to one or more layers of fully connected neurons with ReLU as activation function, before being fed into final output.

3.6 BertMovie

BertMovie is a class wrapper built on top of huggingface transformer model “BertForSequence classification”. BertMovie has its own data tokenization method, called “BertTokenizer”, and its own embedding, Bert Embedding, therefore this model is somewhat independent from all other models I have described so far in terms of data tokenization, vocabulary building and embedding. BertMovie is a transfer learning model, pre-trained on a very large corpus, and then fine tuned on this homework’s dataset. I used both the pretrained “bert base” model (with 724 embedding dimension) and “bert large” model (with 1024 embedding dimensin).

3.7 Ensemble Method - Naive Ensemble

The output from any one of the above models is the last layer’s logits followed by a sigmoid function, and the meaning of such output (with dimension 46, each value ranging from 0 to 1) is the probability of the sample having each of the 46 labels. When I have two outputs from two models, a naive way to combine the two models result is to take an average of the two sigmoid output and use the average to decide the final label. I used this method to produce ensembled results of various combinations of two models. The result is generally improved compared to single-model performance, as will be shown in the performance section.

3.8 Ensemble Method - Multitask Ensemble

Instead of naively taking the average of two model’s sigmoid output, I also try to combine the two models through one final ensemble layer. I concatenate the two output layers of 46 neurons each into a 92 neuron layer, and then produce the final result of 46 output through a fully connected layer. This way the model can dynamically combine the output of two models in a way that’s more sophisticated than simple averaging. To make sure that each individual model is sufficiently trained in addition to the ensemble model, I modify the loss function such that the loss = model1 loss + model 2 loss + ensemble loss. This is a multi-task learning scheme. Surprisingly, the performance of this more “sophisticated” ensemble model is generally worse than naive ensembling, as will be shown in the performance section.

Training Setup

For training the data batch sizes are 32 or 64. After each epoch, I calculate the training loss, validation loss and validation f1 score. I set the “patience” to 10 and stopped training after validation f1 score hasn’t improved for patience number of epochs. The number of epochs it takes to reach patience depends on the model, but generally it ranges from 10-80 epochs. Learning rate also varies between models, with the simpler models using 1e-2 as learning rate and more complex models using learning rate of 1e-5. The learning rate is reduced by a factor of 10 when validation loss hasn’t improved for 10 epochs. I used Adam optimizer. I trained all models on GPU, using my personal PC with two nvidia GTX 1080ti, each of which has a 11Gb memory at 11.3 tensorFLOPS performance.

Performance Comparison

I have tried many models, combination of models and hyperparameters within each model. The following performance table is in no way complete, and I never intended to perform a full grid search of all possible combinations of settings. Therefore I can only highlight a few important things I learnt through my experiments. All results are reported up to two decimal points, since I only want to point out the rough level of performance without getting influenced by variance between runs.

5.1 Baseline

The initial baseline performance of a randomized word embedding followed by logistic regression has a public leaderboard f1 score of **0.675**.

5.2 Ngram results

These are the ngram models using 1-gram or bi-gram with a multilayer perceptron. From this result it's shown that the additional bigram feature doesn't help with performance, therefore I stopped using it.

	Ngram - 1gram	Ngram - 1gram + bigram
MLP	0.78	0.78

5.3 Word embedding results

	Word Embedding - Random Initialization.	Word Embedding - Fasttext Pretrained
MLP	0.76	-
LSTM	0.73	0.75
Bidirectional LSTM	0.74	0.75
Bidirectional GRU	0.79	0.80

These experiments show that pretrained fasttext word embedding works better than randomly initialized embedding, that bidirectional recurrent neural network works better than

uni-directional, and that GRU performs better than LSTM and MLP for word embedding. For subsequent experiments, when I refer to GRU, it means bidirectional GRU.

5.4 Ensemble results

	Single Model result	Ensemble - Average	Ensemble - Multitask
MLP + ngram	0.78	0.82	0.79
GRU + ft embedding	0.80		

This demonstrates that combining different models improves performance by a lot, yet a simple averaging ensemble performs better than a complex multitask ensemble. My understanding of the improvement by ensemble is that very different models learn to use very different aspects of the feature for prediction, and averaging the results from different models combines the strength of from two different perspectives. For example, when one model is very confident about a positive label with p=0.98, and the other model is cautiously judging the same sample to have a negative label with p=0.4, averaging the two probability gives p=0.69, which means the model with more confidence wins. This phenomenon leads to improvement in overall prediction.

5.5 Phrase representation results

This result refers to the feature engineering described in section 2.6, where I replaced the movie names and person names with the concatenation of the names, such that “life is beautiful” is replaced by “life_is_beautiful”. I call the latter “phrase representation”.

	Single Model result	Ensemble - 2 models	Ensemble - 4 models
ngram word	0.78	0.81	0.84
ngram phrase	0.78		
GRU word	0.80	0.81	
GRU phrase	0.80		

This result shows that just by phrase representation alone doesn’t help very much. However, ensembling the result of word and phrase representation improved the final results, probably because the word representation and phrase representation each has their unique advantages, and the advantages are combined after ensembling.

5.6 Character level embedding results

Treating each character in an utterance as an independent unit is promising if there is a lot of data, according to papers such as [1]. However, in my result, the f1 score is extremely poor (around 0.26). This might be improved with hierarchical models where character level information is combined with word level information, however, I didn't delve into this.

5.7 Bert Embedding.

Bert leverages the contextual information [2] in natural language and is shown to outperform context-independent word embedding as such fasttext by a large margin. My result, shown below confirms the advantage of bert. My final leaderboard result of 0.88 is from bert large ensembled with ngram phrase representation model.

	F1 score
Bert Base	0.86
Bert Large	0.87
Bert Large + Ngram Phrase	0.88*

Links and References

- https://github.com/MollyZhang/CSE244_ML_for_NLP_HW1. **Github repo for hw1**
- https://fasttext.cc/docs/en/english-vectors.html. fasttext pretrained embedding.
- https://github.com/huggingface/transformers. pretrained bert models.
- https://medium.com/huggingface/multi-label-text-classification-using-bert-the-mighty-transformer-69714fa3fb3d. bert tutorial.
- https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/ another bert tutorial.
- https://pytorch.org/tutorials/beginner/transformer_tutorial.html. torchtext tutorial.
- [1] Santos, Cicero D., and Bianca Zadrozny. "Learning character-level representations for part-of-speech tagging." *Proceedings of the 31st international conference on machine learning (ICML-14)*. 2014.
- [2] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).