Issue Report Classification

Le Hoang Long[1*]

[1*]School of Information and Communications Technology, Hanoi University of Science and Technology, Dai Co Viet, Hanoi, Vietnam.

Corresponding author(s). E-mail(s): hoanglong1712@gmail.com;

Abstract

NLP-based approaches and tools have been proposed to improve the efficiency of software engineers, processes, and products, by automatically processing natural language artifacts (issues, emails, commits, etc.).
We believe that the availability of accurate tools is becoming increasingly necessary to improve Software Engineering (SE) processes. Two important processes are (i) issue management and prioritization and (ii) code comment classification where developers have to understand, classify, prioritize, assign, etc. incoming issues and code comments reported by end-users and developers.

Keywords: multi-label, classification, issue, GNN, KG, TransR

1 Introduction

The issue report classification competition consists of building and assessing a set of multi-class classification models to classify issue reports as belonging to one category representing the type of information they convey.[1], [2].

We are provided a dataset encompassing 3 thousand labeled issue reports (as bugs, enhancements, and questions) extracted from 5 real open-source projects. We must train, tune, and evaluate multi-class classification models using the provided training and test sets. [3],[4], [5].

2 The architecture and details of the classification models

2.1 Data

In the contest, we use a pre-trained BERT model for multi-label classification. The training dataset is provided in CSV format and consists of six columns: repo, created_at, label, title, and body.

Table 1 Issue table

repo	created_at	label	title	body
facebook/react	some date	bug	some title	some text
facebook/react	some date	feature	some title	some text
facebook/react	some date	question	some title	some text

There are 5 unique values in 'repo' column: 'facebook/react', 'tensorflow/tensorflow', 'microsoft/vscode', 'bitcoin/bitcoin', and 'opencv/opencv'.

[1]There are 3 unique values in 'label' column: 'bug', 'feature', and 'question'.

To pre-process the data, we combine 2 columns "title, and body" into a new column named "data" and generate a list of labels for each instance "'facebook/react', 'tensorflow/tensorflow', 'microsoft/vscode', 'bitcoin/bitcoin', 'opencv/opencv', 'bug', 'feature', and 'question'", which can be represented in binary format (1 for presence, 0 for absence)

Table 2 New issue table

created_at	data	target_list
2023-08-26 06:33:37	[DevTools Bug] Cannot add node "1"...	[1, 0, 0, 1, 0, 0, 0, 0]
2023-07-28 05:16:12	[DevTools Bug]: Devtools extension ...	[1, 0, 0, 1, 0, 0, 0, 0]
2023-07-13 21:58:31	[DevTools Bug]: Deprecated __REACT...	[1, 0, 0, 1, 0, 0, 0, 0]
2023-06-14 02:31:20	[DevTools Bug] Cannot remove node "0"...	[1, 0, 0, 1, 0, 0, 0, 0]
2023-06-03 11:29:44	[DevTools Bug] Cannot remove node "103"...	[1, 0, 0, 1, 0, 0, 0, 0]

2.2 Model

```
class BERTClass(torch.nn.Module):
    def __init__(self):
        super(BERTClass, self).__init__()
        self.l1 = transformers.BertModel.from_pretrained('bert-base-uncased',
        return_dict=False)
        self.l2 = torch.nn.Dropout(0.3)
```

```
        self.l3 = torch.nn.Linear(768, 6)

    def forward(self, ids, mask, token_type_ids):
        _, output_1= self.l1(ids, attention_mask = mask, token_type_ids = token_type_ids)
        output_2 = self.l2(output_1)
        output = self.l3(output_2)
        return output
BERTClass(
  (l1): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
```

```
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (activation): Tanh()
      )
    )
    (l2): Dropout(p=0.3, inplace=False)
    (l3): Linear(in_features=768, out_features=8, bias=True)
)
```

3 The procedure used to pre-process the data

We use pandas library to pre-process the data,

```python
import pandas as pd
import numpy as np

def add_collum(src_name, col_name, frame):
  frame[col_name] = np.where(frame[src_name] == col_name, 1, 0)
  return frame

def preprocessing(csv_name):
  df = pd.read_csv(csv_name)
  for item in df.repo.unique():
    df = add_collum('repo', item, df)
  for item in df.label.unique():
    df = add_collum('label', item, df)
  df['data'] = df.title + ' ' + df.body

  ndf = df.drop(['title', 'body', 'repo', 'label'], axis=1)

  ndf['target_list'] = ndf[['bug', 'feature', 'question',
                            'facebook/react', 'tensorflow/tensorflow',
                            'microsoft/vscode', 'bitcoin/bitcoin',
                            'opencv/opencv']].values.tolist()
  df2 = ndf.drop(['bug', 'feature', 'question',
                            'facebook/react', 'tensorflow/tensorflow',
                            'microsoft/vscode', 'bitcoin/bitcoin',
                            'opencv/opencv'], axis=1)
  return df2
```

Table 3 Old table

repo	created_at	label	title	body
facebook/react	2023-07-28 05:16:12	bug	some title	some text
facebook/react	2023-07-28 05:16:12	feature	some title	some text
facebook/react	2023-07-28 05:16:12	question	some title	some text

Table 4 New table

created_at	data	target_list
2023-08-26 06:33:37	[DevTools Bug] Cannot add node "1"...	[1, 0, 0, 1, 0, 0, 0, 0]
2023-07-28 05:16:12	[DevTools Bug]: Devtools extension ...	[1, 0, 0, 1, 0, 0, 0, 0]
2023-07-13 21:58:31	[DevTools Bug]: Deprecated __REACT...	[1, 0, 0, 1, 0, 0, 0, 0]
2023-06-14 02:31:20	[DevTools Bug] Cannot remove node "0"...	[1, 0, 0, 1, 0, 0, 0, 0]
2023-06-03 11:29:44	[DevTools Bug] Cannot remove node "103"...	[1, 0, 0, 1, 0, 0, 0, 0]

4 The procedure used to tune the classifiers on the training sets

4.1 Python source code

```
MAX_LEN = 250
TRAIN_BATCH_SIZE = 64
VALID_BATCH_SIZE = 64
EPOCHS = 4
LEARNING_RATE = 1e-06 * 5
# https://stackoverflow.com/questions/65082243/dropout-argument-input-position-1-must
#-be-tensor-not-str-when-using-bert
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', return_dict=False)

def loss_fn(outputs, targets):
    return torch.nn.BCEWithLogitsLoss()(outputs, targets)

optimizer = torch.optim.Adam(params =  model.parameters(), lr=LEARNING_RATE)

def train_model(start_epochs,  n_epochs, valid_loss_min_input,
                training_loader, validation_loader, model,
                optimizer, checkpoint_path, best_model_path):

  # initialize tracker for minimum validation loss
  valid_loss_min = valid_loss_min_input

  for epoch in range(start_epochs, n_epochs+1):
    model.train()
    print('############# Epoch {}: Training Start   #############'.format(epoch))
    for batch_idx, data in enumerate(training_loader):

        ids = data['ids'].to(device, dtype = torch.long)
        mask = data['mask'].to(device, dtype = torch.long)
        token_type_ids = data['token_type_ids'].to(device, dtype = torch.long)
        targets = data['targets'].to(device, dtype = torch.float)

        outputs = model(ids, mask, token_type_ids)
```

```
        optimizer.zero_grad()
        loss = loss_fn(outputs, targets)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print('############# Epoch {}: Training End     #############'.format(epoch))

  return model

checkpoint_path = 'current_checkpoint.pt'
best_model = 'best_model.pt'
trained_model = train_model(1, 30, np.Inf, training_loader, validation_loader, model,
                      optimizer,checkpoint_path,best_model)
```

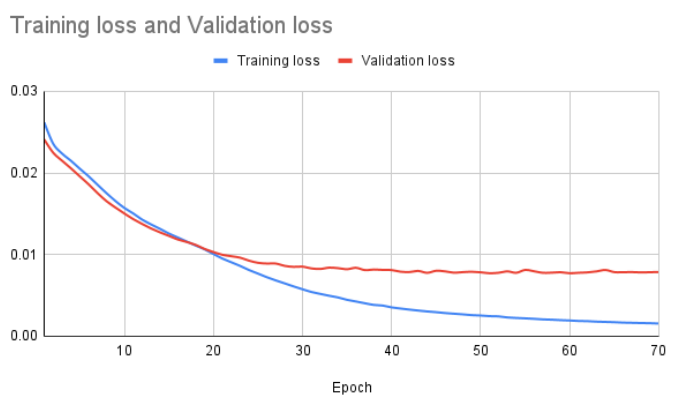
4.2 Explanation

The loss function is BCEWithLogitsLoss which is a suitable solution to multi-label classification projects. BCEWithLogitsLoss combines a Sigmoid layer and the BCELoss in one single class. This version is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.

"TRAIN_BATCH_SIZE = 64" is a suitable constant, it helps to lower the value of loss function. We had used 32, which is a common setting, and the batch wasn't big enough to extract all required features.

The setting "LEARNING_RATE = 1e-06 * 5" performs better in the project compared to the more common "LEARNING_RATE = 1e-05."

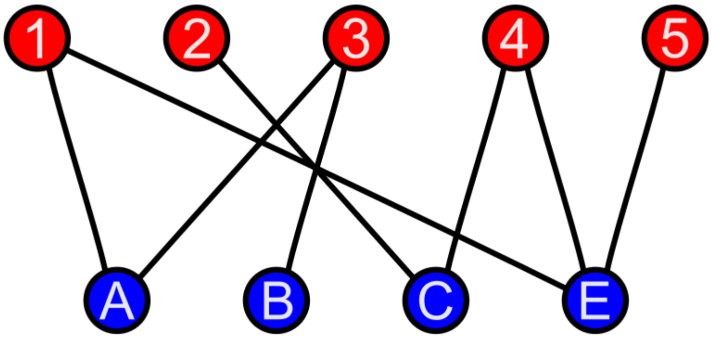
5 The results of classifiers on the test sets



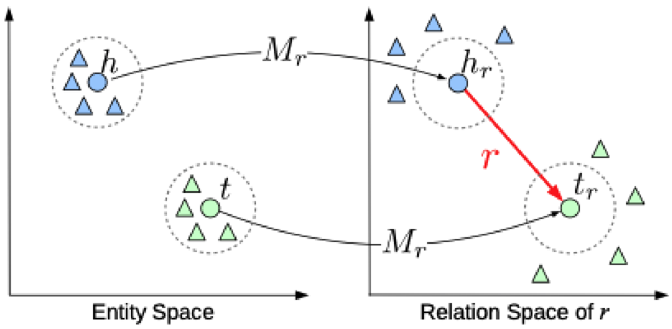
6 Future work

The current model functions, but it struggles with large-scale datasets. Training it on even a small dataset requires significant time and computing resources, making it impractical for real-world classification tasks.

Knowledge graph is an effective approach in this scenario. We could see that the dataset consists of two types of items: labels and data. The data is in text format, and there are eight different labels. We can construct a bipartite graph (or bigraph) based on the dataset. Bigraph is a graph whose vertices can be divided into two disjoint and independent sets and, that is, every edge connects a vertex into one in.



Knowledge graph completion focuses on link prediction between entities. In future work, we plan to explore knowledge graph embeddings. We are going to use TransR to create separate embeddings for entities and relations in their respective spaces. Then, we learn these embeddings by projecting entities from their space into the corresponding relation space and establishing translations between the projected entities [6], [7].



We apply a new method that models entities and relations in separate spaces specifically, an entity space and multiple relation-specific spaces. This approach performs translations within the corresponding relation space.

The fundamental concept of TransR is depicted in the figure above. For each triple (h, r, t), the entities in the entity space are first projected into the r-relation space as h_r and t_r using the operation M_r. This leads to the approximation $h_r + r_r \approx t_r$. The relation-specific projection brings together head and tail entities that share the relation (represented as colored circles) while distancing them from those that do not (represented as colored triangles).

In TransR, for each triple (h, r, t), the entity embeddings are represented as $h, t \in R^k$, while the relation embeddings are denoted as $r \in R^d$. It's important to note that the dimensions of the entity and relation embeddings do not have to be the same, meaning $k \neq d$.

For each relation r, we define a projection matrix $M_r \in R^{k \times d}$ that projects entities from the entity space into the relation space. Using this mapping matrix, we define the projected vectors of the entities as follows:

$$h_r = hM_r, t_r = tM_r$$

The score function is correspondingly defined as

$$f_r(h,t) = \parallel h_r + r - t_r \parallel_2^2$$

We define the following margin-based score function as the objective for training

$$L = \sum_{(h,r,t)\in S} \sum_{(h',r',t')\in S'} max(0, f_r(h,t) + \gamma - f_r(h',t'))$$

where max(x, y) aims to get the maximum between x and y, γ is the margin, S is the set of correct triples, and S' is the set of incorrect triples.

References

[1] Kallis, R., Colavito, G., Al-Kaswan, A., Pascarella, L., Chaparro, O., Rani, P.: The nlbse'24 tool competition. In: Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24) (2024)

[2] Al-Kaswan, A., Izadi, M., Van Deursen, A.: Stacc: Code comment classification using sentencetransformers. In: 2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE), pp. 28–31 (2023)

[3] Kallis, R., Di Sorbo, A., Canfora, G., Panichella, S.: Predicting issue types on github. Science of Computer Programming **205**, 102598 (2021) https://doi.org/10.1016/j.scico.2020.102598

[4] Kallis, R., Di Sorbo, A., Canfora, G., Panichella, S.: Ticket tagger: Machine learning driven issue classification. In: 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019, pp. 406–409. IEEE, ??? (2019). https://doi.org/10.1109/ICSME.2019.00070

[5] Pascarella, L., Bacchelli, A.: Classifying code comments in java open-source software systems. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR) (2017). IEEE

[6] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Burges, C.J., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 26. Curran Associates, Inc., ??? (2013). https://proceedings.neurips.cc/paper$_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf$

[7] Yankai Lin, M.S.Y.L.X.Z. Zhiyuan Liu: In: Learning Entity and Relation Embeddings for Knowledge Graph Completion. https://linyankai.github.io/publications/aaai2015_transr.pdf