CS598 Project Final 2023

Chhavi Jain and Rathiv Bal
{chhavi2, rbal2}@illinois.edu

Group ID: 126
Paper ID: 104

Presentation Link: youtube, slides(please click on raw)
Code Link: github link

1. Introduction

This paper is solving the problem of unstructured clinical notes labelling with ICD codes.

Due to the unstructured nature of clinical notes, providers often employ trained staff and/or third-party vendors to help assign diagnostic codes using coding systems such as the International Classification of Diseases (ICD). However, manual assignment of ICD codes is problematic:

1. It is both time consuming and error-prone, with only 60-80% of the assigned codes reflecting actual patient diagnoses.
2. A significant portion of code assignment results in misjudged severity of conditions and code omissions
3. For healthcare providers, billing and coding errors may not only lead to loss of revenue and claim denials, but also federal penalties for erroneous Medicare and Medicaid claims.

Thus, there is a clear need for reliable automated classification of unstructured clinical notes. The paper "Classifying Unstructured Clinical Notes via Automatic Weak Supervision" proposes a low-cost solution that has the potential to address some of the identified pressing issues with accessibility to affordable yet accurate automated disease coding tools.

2. Scope of reproducibility

This paper introduces a new approach to efficiently acquire interpretable weak supervision sources from readily available text to learn effective text classifiers without the need for human-labeled data. This paper gives KeyClass framework for automated text classification, which combines weak supervision, data programming, and self-training techniques to automatically generate high-quality training labels for text classification tasks.

Our Scope is to use KeyClass approach explained above with movie database to verify the claim in paper and check if KeyClass approach can give highly accurate classifier for movie data on review classification task(recommended or not recommended movie review).

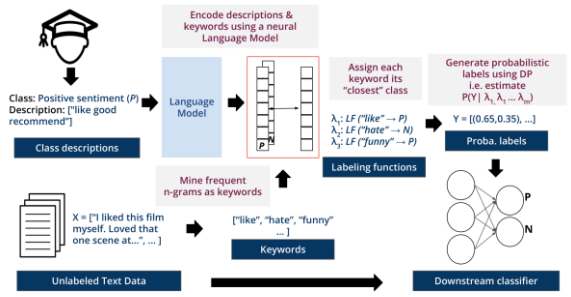
2.1 Addressed claims from the original paper

We are testing following claims from the paper

- Claim 1: Pre-trained language models can efficiently and effectively inform weakly supervised models for text classification
- Claim 2: Self-training improves downstream classifier performance, especially when classifiers are initially trained on a subset of the training data
- Claim 3: Data programming performs on par with simple majority vote when relying on a large number of automatically generated weak supervision sources of similar quality.
- Claim 4: Keywords are excellent sources of weak supervision

3. Methodology

Overview of methodology:

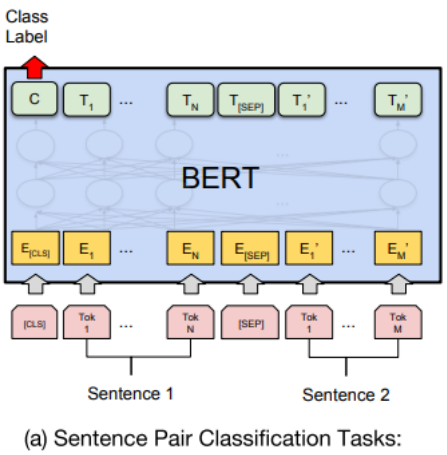


From only class descriptions, KeyClass classifies documents without access to any labeled data.

It automatically creates interpretable labeling functions (LFs) by extracting frequent keywords and phrases that are highly indicative of a particular class from the unlabeled text using a pre-trained language model. It then uses these LFs along with Data Programming (DP) to generate probabilistic labels for training data, which are used to train a downstream classifier

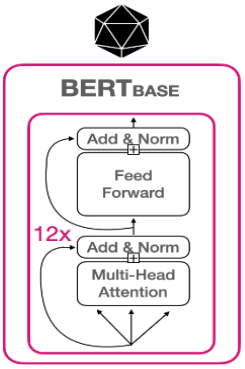
3.1 Model descriptions

In the original paper, KeyClass framework leverages the general linguistic knowledge stored within pre-trained neural language models such as Bidirectional Encoder Representations from Transformers **(BERT)**[4], to map each keyword to the most semantically related category description. To create a labeling function, KeyClass simply assigns a keyword to its closest category as measured by the cosine similarity between their embeddings. In order to ensure equal representation of all classes, KeyClass sub-samples the top-k labeling functions per class, ordering them by cosine similarity. While theoretically data programming benefits from as many labeling functions as possible, the sampling is required due to computational and space constraints.



(a) Sentence Pair Classification Tasks:

BERT Size & Architecture



110M Parameters

Model architecture:

1. In paper model used BERT base and similar models (we used specifically sentence-transformers[3]/paraphrase-mpnet-base-v2 model: that maps sentences & paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks[2]:
2. Parameters: 110M parameters
3. Transformer Layers (Number of transformer blocks): 12
4. Hidden size (Layers of mathematical functions, located between the input and output, that assign weights (to words) to produce a desired result): 768
5. Attention Heads: 12
6. Processing: 1 GPU
7. Length of training: 0.5 day

3.2 Data descriptions

We have used Imdb movie dataset using the following command

*wget --load-cookies /tmp/cookies.txt "https://drive.google.com/a/illinois.edu/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate 'https://drive.google.com/a/illinois.edu/uc?export=download&id=1c8X_Ooth2fQleCVz2gCXlOd3-zzE9Mws' -O- | sed -rn 's/.*confirm=([0-9A-Za-z_]+).*/\1\n/p')&id=1c8X_Ooth2fQleCVz2gCXlOd3-zzE9Mws" -O imdb.zip && rm -rf /tmp/cookies.txt*

Data description - IMDB movie dataset is a large d ataset containing information on over 85,000 mo vies and TV shows, dataset is 162 MB

- Size of the data: 25000 samples
- Len keywords 600

Find Relevant Keywords / Encoding the Dataset
Probabilistic labeled dataset generated by model training by using Imdb dataset

Assigned category:
Not recommend ['abominable' 'abomination' 'absolute worst' 'absolutely awful' 'absolutely terrible' 'abuse' 'abused' 'abusive' 'abysmal' 'acting horrible' 'acting poor' 'acting terrible', 'writing bad'] and so on.

Recommend ['actually good' 'actually like' 'admirable' 'admirably' 'best' 'best best' 'best ever' 'best one' 'best parts' 'best performance' 'best show' 'best thing' 'best things' 'better ones' 'certainly good' 'commendable' 'damn good'] and so on

Data statistics
Size of data: (25000, 768)
Size of labels: 25000
Training class distribution (ground truth): [0.5 0.5]
Training class distribution (label model predictions): [0.35656 0.64344]

3.3 Hyperparameters

After obtaining a probabilistically labeled training dataset, KeyClass can train any downstream classifier using rich document feature representations provided by the neural language model. Instead of using all the automatically labeled documents, KeyClass initially trains the downstream classifier using top-k documents with the most confident label estimates only. We used following hyperparameters:

activation: torch.nn.LeakyReLU()

average: weighted

base_encoder: paraphrase-mpnet-base-v2

criterion: torch.nn.CrossEntropyLoss(reduction='none')

dataset: imdb

device: cuda

end_model_batch_size: 128

end_model_epochs: 20

end_model_lr: 1e-4

end_model_patience: 3

end_model_weight_decay: 1e-4

batch_sizes:- variable(reducing by

2 for fine tuning steps started with 512)

label_model: data_programming

label_model_lr: 0.01

label_model_n_epochs: 100

max_num: 7000

min_df: 0.001

model_path: ../models/imdb/

n_bootstrap: 100

n_classes: 2

n_jobs: 10

normalize_embeddings: false

preds_path: ../results/imdb/

q_update_interval: 50

results_path: ../results/imdb/

self_train_batch_size: 8

self_train_lr: 1e-6

self_train_patience: 3

self_train_thresh: 1-2e-3

self_train_weight_decay: 1e-4

show_progress_bar: true

target_0: negative, hate, expensive, bad, poor, broke, waste, horrible, would not recommend

target_1: good, positive, excellent, amazing, love, fine, good quality, would recommend

topk: 300

use_custom_encoder: false

use_noise_aware_loss: true

3.4 Implementation

We leveraged code from original paper authors at [2]. We updated our code with steps to download MIMIC-III dataset. Added Jupyter notebook for reproducing paper results.

Also added configuration files corresponding to our experiments. Added results from our experiments in assets folder. Also, added information about classification results visualization using what-if tool as well added tutorial about tool usage.

Also added code related to our ablation study in corresponding folder to explore prompting approach for improving performance on low prevalence classes using "Knowledge Injected Prompt Based Fine-tuning for Multi-label Few-shot ICD Coding" . Here is our code link

3.5 Computational requirements

Compute resource estimates

For IMdb dataset experiments. We used google colab notebook standard GPU , we used closed to half a day of training

Memory estimates

For a 110 million parameter model and a 16MB dataset, assuming a batch size of 1, we can estimate

the GPU memory requirement as follows:

- Model memory: The model's memory requirement can be estimated by multiplying the number of parameters by the size of each parameter. For example, if each parameter is represented using 32-bit floating-point numbers (float32), then the model's memory requirement would be:

110 million * 32 bits / 8 bits per byte = 440 MB

Total dataset size 16MB and it has 25000 samples of data each sample is 16MB/25000 = 0.64 KB

With a batch size of 512, the amount of memory required to store one batch of data would be:

512 * 0.64 KB = 0.32 MB

We are loading 10 batched of data at a time

Assuming we can load and process 10 batches of data at a time, then the total amount of GPU memory required would be:

- Model memory: 440 MB (calculated as before)
- One batch of data memory: 0.32 MB (calculated as before)
- Additional memory overhead: 1-2x model memory, which is 440 MB - 880 MB

The total memory requirement would be approximately:

(440 MB + 10 * 0.32 MB) * 2 = 443.2 * 2 MB

~ 900MB

So, we can estimate that a 110 million parameter model and a dataset of 25,000 samples each 0.64 KB in size, with a batch size of 512, would require approximately 900 MB of GPU memory. So, with all the overhead we will keep an estimated 800MB memory in GPU. From verification of jupyter notebook code memory size we used close to 600MB. MIMIC-III dataset was the largest with 6.2GB data and 26 relational tables so we had to use pruning and batching techniques to fit the data.

4 Results

We support the claims listed in section 2.1

Our results mostly match the results in paper however we are mostly off by 1st or second decimal place in accuracy numbers our results are little worse within +- 0.9 in most cases. With more training iterations and lower lr accuracy might improve little bit but our training loops converged at below numbers.

4.1 Result 1

Results on experiments mentioned in paper

Database: dbpedia

	mean	std
Accuracy	0.930212	0.000846
Precision	0.934717	0.000774
Recall	0.930272	0.000846

Database Imdb

	mean	std
Accuracy	0.870569	0.002177
Precision	0.870046	0.002216
Recall	0.870569	0.002177

Database MIMIC

	mean	std
Accuracy	0.890205	0.000364
Precision	0.507333	0.000275
Recall	0.625217	0.000364

4.2 Result 2

For Imdb dataset

While deciding class for labels during training time

- Confidence of least confident data point of class 0: 0.9118951704039087
- Confidence of least confident data point of class 1: 0.9999157388534687

For MIMIC dataset

Model performance on MIMIC

Category Name KeyClass

- Infectious & parasitic 0.465
- Neoplasms 0.027
- Endocrine, nutritional and metabolic 0.843
- Blood & blood-forming organs 0.609
- Mental disorders 0.483 0.384
- Nervous system 0.312
- Sense organs 0.006
- Circulatory system 0.915
- Respiratory system 0.676
- Digestive system 0.608
- Genitourinary system 0.612
- Pregnancy & childbirth complications 0.000
- Skin & subcutaneous tissue 0.004
- Musculoskeletal system & connective

tissue0.078

- Congenital anomalies 0.016
- Perinatal period conditions 0.000
- Injury and poisoning 0.599
- External causes of injury 0.604
- Supplementary 0.826

4.3 Additional results not present in the original paper

We saw that with MIMIC-III dataset, some classes had very low accuracy and although our results were within +- 0.9 of original paper but we wanted to explore. After discussion with original paper authors we tried prompting approach and results improved for low prevalence classes here are results with Knowledge Injected Prompt Based Fine-tuning for Multi-label Few-shot ICD Coding Automatic International Classification of Diseases (ICD) coding aims to assign multiple ICD codes to a medical note with average length of 3,000+ tokens.

Model	AUC	F1	Precision
KEPTLongformer	91.07	93.45	64.82

5 Discussion

The original paper was reproducible. The evidence we got from running the code supports the claims of the paper. The strengths of the approach are that this approach reduces the manual effort quite a bit for very tedious and important task of unstructured clinical notes to ICD label assignment problem .
The approach assumes that ICD code description will be detailed enough to drive label keywords from them . And the assumption seems valid to us.

The weaknesses this approach is explain ability it does not give any reason why in some cases it does not work for example why accuracy is not 100% , and which exact cases should be sent for human review. Even 90% accuracy is not good enough if we don't know which clinical notes are not assigned correct ICD codes. Adding some visualization and explain-ability to model results will help the overall approach gain trust and help it got widely adapted in real world unstructured clinical notes to ICD code assignment problem. We have added what-if tool usage tool to our code repository.

5.1 What was easy

It was easy to run the author's code for most part, some notebooks assume that data is present, and path is present on the system where it is running achieving that with jupyter notebook runtime environment took some time. The code had extensive API documentation and a lot of examples that matched experiments in papers. It's a very well documented code in general and made it very good exercise to run and understand it.

5.2 What was difficult

We had to wait for sometime for MIMIC III dataset and also the results on MIMIC-III were more difficult to reproduce because of large size of dataset . We had to spend sometime in data pruning , parallelization, batching and hyper-parameter tuning techniques.

5.3 Recommendations for reproducibility

We have jupyter notebook at link for reproducing model results. Also we documented all the steps clearly in readme section of our code repo. Results of paper are reproducible with minor change in accuracy +-0.9

6 Communication with original authors

Low-support categories, i.e., categories with low prevalence in data performed less accurately. We tried adjusting the classifier's algorithm for loss function to account for the imbalance but results couldn't be improved. We tried tweaking hyperparameters as well but it was clear that with given approach low prevalence classes will not have higher accuracy

We discussed with original authors of paper as well. And explored prompting approach to improve results on low prevalence classes. And yes results did improve. We also discussed using chatgpt type of model for generating more data but adding data through more clinical datasets or synthetic data generation methods like SMOTE sounded preferable.

7 References

[1] https://arxiv.org/abs/1908.10084
[2] https://github.com/autonlab/KeyClass
[3] https://www.sbert.net/
[4]https://github.com/autonlab/KeyClass/blob/5e75c756133e45f697a629cfb411e2debbadd039/tutorials/#references
[5] https://arxiv.org/pdf/2210.03304v2.pdf
[6]https://github.com/whaleloops/KEPT