Keyclass Ablation Study

By:

Pascal Adhikary (pascala2)

Rustom Ichhaporia (rustomi2)

Video Presentation: https://youtu.be/NudHNdBaJFc

GitHub: https://github.com/rustom/cs598-dlh-team-17/tree/main

Introduction

Background of the Problem

The problem at hand is the classification of unstructured clinical notes. This is a crucial task in healthcare informatics, with applications ranging from disease prediction, readmission prediction, mortality prediction, to feature engineering and data processing. The importance of solving this problem lies in its potential to improve patient care and outcomes, streamline healthcare operations, and contribute to medical research.

However, this task is challenging due to the unstructured and complex nature of clinical notes, which often contain medical jargon, abbreviations, and patient-specific information. State-of-the-art methods often rely on manual diagnostic coding or supervised learning models, which require large amounts of labeled data. These methods can be time-consuming, expensive, and prone to errors.

Paper Explanation

The paper ‘Classifying Unstructured Clinical Notes via Automatic Weak Supervision’ proposes a solution to this problem with KeyClass, a weakly-supervised text classification framework. The innovation of this method lies in its ability to learn a text classification model from class label descriptions alone, eliminating the need for manually tagged documents.

KeyClass leverages pre-trained language models and data programming frameworks to assign code labels to specific texts. During the classification process, it creates interpretable heuristics based on keywords extracted from the available text data. This approach allows KeyClass to adapt to classification tasks in other highly specialized domains through the use of domain-specific language models.

The effectiveness of KeyClass is demonstrated in its own metrics, showing superior performance compared to other weakly supervised text classification architectures. This underscores the efficiency and versatility of KeyClass.

Contribution to the Research Regime

The contribution of this paper to the research regime is significant. By introducing a weakly-supervised framework that eliminates the need for manually tagged documents, it addresses the main challenges associated with the classification of unstructured clinical notes. This not only streamlines the process of diagnostic coding but also opens up possibilities for further research and applications in healthcare informatics.

Scope of Reproducibility:

1. **Hypothesis 1**: KeyClass performs on par with the fully supervised baseline FasTag (Venkataramam et al.) on the MIMIC-III ICD-9 code assignment problem in terms of recall, precision, and f1 metrics.
2. **Hypothesis 2**: The KeyClass model is able to extract frequent keywords and phrases that are highly indicative of a particular class from the unlabeled text using a pre-trained language model.
3. **Ablation**: Self-training vs no self-training improves downstream classifier performance.

Methodology

This methodology is the core of your project. It consists of run-able codes with necessary annotations to show the expeiment you executed for testing the hypotheses.

The methodology at least contains two subsections **data** and **model** in your experiment.

1. **Identifying Class Descriptions**: The first step in the unsupervised classification process employed by KeyClass involves utilizing natural language descriptions corresponding to each class in a dataset.
2. **Extraction of Pertinent Keywords**: In this step, the KeyClass framework identifies keyphrases of high relevance for each class. This is achieved by extracting frequent n-grams and associating them with the category description that they are most semantically related to, as determined by a pre-trained language model.
3. **Probabilistic Labeling of Data**: This step involves the construction of a labeling function vote matrix, which is then used to generate probabilistic labels for all the documents in the training set.
4. **Training the Downstream Classifier**: The final step involves training the downstream classifier using the rich feature representations provided by the neural language model. The initial training is performed on the top 'k' documents, where 'k' is a parameter set by the user representing the most confident documents. Following this, self-training is performed on the entire dataset as a final step of refinement.

Environment

Python version: 3.10.12

Dependencies:

- MIMIC-III dataset
- `pandas` / `numpy`
- `KeyClass` and related code, which can be found on GitHub
- `FasTag` for optional training data preprocessing, which can be found on GitHub

Original Paper:

Chufan Gao, Mononito Goswami, Jieshi Chen, and Artur Dubrawski. 2022. Classifying Unstructured Clinical Notes via Automatic Weak Supervision. Machine Learning for Healthcare Conference (2022).

https://arxiv.org/abs/2206.12088

In [ ]:
```
# Check your python version
!python -V
```

Data

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
```

A summary of the 19 categories that KeyClass attempts to classify and their prevalence in the dataset is pasted below.

Category	Prevalence	Description
1	0.1675	infections, parasitic
2	0.04375	neoplasms
3	0.02875	endocrine, nutritional, metabolic
4	0.04	blood, blood-forming organs
5	0.015	mental disorders
6	0.0175	nervous system
7	0.05	sense organs
8	0.0125	circulatory system
9	0.025	respiratory system
10	0.02875	digestive system
11	0.015	genitourinary system
12	0.02	pregnancy, childbirth complications
13	0.04375	skin, subcutaneous tissue
14	0.0125	musculoskeletal system, connective tissue
15	0.04125	congenital anomalies
16	0.00875	perinatal period conditions
17	0.225	injury and poisoning
18	0.12125	external causes of injury
19	0.08375	supplementary

In [ ]:
```
# Because of the large size of the data, the following code was run locally to
# generate a small subsample of the data that was uploaded to Google Drive
# for demonstration purposes.

raw_data_generation = '''
base_path = '/Users/rustomichhaporia/GitHub/KeyClass/team_17/data/'

diagnoses_icd = pd.read_csv(base_path + 'DIAGNOSES_ICD.csv', usecols=['HADM_ID', 'ICD9_C
note_events = pd.read_csv(base_path + 'NOTEEVENTS.csv', usecols=['HADM_ID', 'CATEGORY',
note_events = note_events[note_events['CATEGORY'] == 'Discharge summary']

output = pd.merge(diagnoses_icd, note_events, on='HADM_ID', how='inner').drop_duplicates
output.sample(1000).to_csv(base_path + 'mimic_subset.csv', index=False)
'''
```

In [ ]:

```
# dir and function to load raw data
raw_data_dir = '/content/drive/MyDrive/mimic_subset.csv'

def load_raw_data(raw_data_dir):
    return pd.read_csv(raw_data_dir)

raw_data = load_raw_data(raw_data_dir)

# calculate statistics
def calculate_stats(raw_data):
  # implement this function to calculate the statistics
  # it is encouraged to print out the results
    print(raw_data.shape)
    print(raw_data.isna().count())

def process_data(raw_data):
    # implement this function to process the data as you need
    def get_high_level_code(raw_code):
        high_level_codes = {
            0: (0, 139),
            1: (140, 239),
            2: (240, 279),
            3: (280, 289),
            4: (290, 319),
            5: (320, 389),
            6: (390, 459),
            7: (460, 519),
            8: (520, 579),
            9: (580, 629),
            10: (630, 679),
            11: (680, 709),
            12: (710, 739),
            13: (740, 759),
            14: (760, 779),
            15: (780, 799),
            16: (800, 999),
            17: (1000, 2000),
            18: (2001, 3000),
        }

        high_level_code = 0
        if raw_code.startswith('E'):
                high_level_code = 1000
        elif raw_code.startswith('V'):
            high_level_code = 1000
        elif raw_code.startswith('U'):
            high_level_code = 2001
        high_level_code = int(raw_code[1:])

        for high_level, (bottom, top) in high_level_codes.items():
            if bottom <= high_level_code <= top:
                return high_level
    raw_data['ICD9_CODE'] = raw_data['ICD9_CODE'].apply(get_high_level_code)
    return raw_data.dropna()



processed_data = process_data(raw_data)
```

In [ ]:
```
!mkdir '/content/drive/MyDrive/mimic-iii/'
base_path = '/content/drive/MyDrive/mimic-iii/'
processed_data.ICD9_CODE.value_counts().shape
processed_data['TEXT'] = processed_data['TEXT'].str.replace('\n', ' ').replace('\r', '')
processed_data.to_csv(base_path + 'mimic_subset_processed.csv', index=False)

from sklearn.model_selection import train_test_split

X = processed_data['TEXT']
y = processed_data['ICD9_CODE']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

X_train.to_csv(base_path + 'train.txt', index=False, header=False)
X_test.to_csv(base_path + 'test.txt', index=False, header=False)
y_train.astype(int).to_csv(base_path + 'train_labels.txt', index=False, header=False)
y_test.astype(int).to_csv(base_path + 'test_labels.txt', index=False, header=False)
```

Model

In [ ]:
```
config_custom = '''
activation: torch.nn.LeakyReLU()
average: weighted
base_encoder: paraphrase-mpnet-base-v2
criterion: torch.nn.CrossEntropyLoss(reduction='none')
data_path: /content/drive/MyDrive/
dataset: mimic-iii
device: cpu
end_model_batch_size: 128
end_model_epochs: 20
end_model_lr: 1e-4
end_model_patience: 3
end_model_weight_decay: 1e-4
h_sizes:
- 768
- 19
label_model: data_programming
label_model_lr: 0.01
label_model_n_epochs: 5
max_num: 2000
min_df: 0.001
model_path: ../models/mimic-iii/
n_bootstrap: 100
n_classes: 19
n_jobs: 10
ngram_range: !!python/tuple
- 1
- 3
normalize_embeddings: false
preds_path: ../results/mimic-iii/
q_update_interval: 50
results_path: ../results/mimic-iii/
self_train_batch_size: 2
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_00: infections, parasitic
target_01: neoplasms
target_02: endocrine, nutritional, metabolic
target_03: blood, blood-forming organs
target_04: mental disorders
target_05: nervous system
```

```
target_06: sense organs
target_07: circulatory system
target_08: respiratory system
target_09: digestive system
target_10: genitourinary system
target_11: pregnancy, childbirth complications
target_12: skin, subcutaneous tissue
target_13: musculoskeletal system, connective tissue
target_14: congenital anomalies
target_15: perinatal period conditions
target_16: injury and poisoning
target_17: external causes of injury
target_18: supplementary
topk: 100
use_custom_encoder: false
use_noise_aware_loss: true
'''

with open('/content/drive/MyDrive/config_custom.yml', 'w') as text_file:
  text_file.write(config_custom)

driver_text = '''
pip install snorkel sentence-transformers pyhealth transformers

BASE_PATH="/content/drive/MyDrive/"

cd -- "$BASE_PATH"
git clone https://github.com/autonlab/KeyClass
cd KeyClass/scripts/
python run_all.py --config /content/drive/MyDrive/config_custom.yml
'''
with open('/content/drive/MyDrive/driver_text.sh', 'w') as text_file:
    text_file.write(driver_text)

!bash /content/drive/MyDrive/driver_text.sh
# !cat /content/drive/MyDrive/config_custom.yml
```

In [ ]:

```
# # Based on keyclass pretrained model test data tutorial code
# Again, not feasible to run in the Colab notebook

# end_model_path='../models/rustom_model_1.pth'
# end_model_self_trained_path='../models/rustom_manual_2.pth'

# args = utils.Parser(config_file_path=config_file_path).parse()

# random_seed = random_seed
# torch.manual_seed(random_seed)
# np.random.seed(random_seed)

# X_train_embed_masked, y_train_lm_masked, y_train_masked, \
#       X_test_embed, y_test, training_labels_present, \
#       sample_weights_masked, proba_preds_masked = train_downstream_model.load_data(arg

# model = torch.load(end_model_path)

# end_model_preds_train = model.predict_proba(torch.from_numpy(X_train_embed_masked), ba
# end_model_preds_test = model.predict_proba(torch.from_numpy(X_test_embed), batch_size=

# if training_labels_present:
#       training_metrics_with_gt = utils.compute_metrics(y_preds=np.argmax(end_model_pre
#
#
#       print('training_metrics_with_gt', training_metrics_with_gt)

# training_metrics_with_lm = utils.compute_metrics(y_preds=np.argmax(end_model_preds_tra
#
#
# print('training_metrics_with_lm', training_metrics_with_lm)

# testing_metrics = utils.compute_metrics_bootstrap(y_preds=np.argmax(end_model_preds_te
#
#
#
#
# print('testing_metrics', testing_metrics)


# # Downstream classifier
# X_train_text = utils.fetch_data(dataset=args['dataset'], path=args['data_path'], split
# X_test_text = utils.fetch_data(dataset=args['dataset'], path=args['data_path'], split=

# model = torch.load(end_model_self_trained_path)

# end_model_preds_test = model.predict_proba(X_test_text, batch_size=args['self_train_ba


# testing_metrics = utils.compute_metrics_bootstrap(y_preds=np.argmax(end_model_preds_te
#
#
#
#
# print('testing_metrics after self train', testing_metrics)
```

Training

Hyperparams

- Batch size: 4
- Train learning rate: 1e-4
- Hidden layer sizes: found in config file
- Q update interval: 50

Computational requirements

- Attempted to run locally with MacBook Pro 2019 Intel i7 with 16GB memory
- Colab specs variable
- 20 training epochs

Results

Table of results

Self training / fine-tuning downstream classifier

id	Precision	Recall	F1
0	0.938	0.872	0.973
1	0.918	0.798	0.833
2	0.513	0.907	0.521
3	0.440	0.997	0.478

4	0.646	0.559	0.678
5	0.599	1.004	0.659
6	0.839	0.407	0.997
7	0.825	0.890	0.417
8	0.844	0.126	0.508
9	0.491	0.516	0.682
10	0.898	1.022	1.042
11	0.590	0.656	0.691
12	0.197	0.655	0.223
13	0.596	0.499	0.423
14	0.725	0.703	0.863
15	0.731	0.226	-0.011
16	0.782	0.245	0.065
17	0.606	0.722	0.438
18	0.667	0.531	0.521

Ablation: no self training

id	Precision	Recall	F1
0	0.051	0.294	0.338
1	0.101	0.271	0.370
2	0.085	0.711	0.708
3	0.521	0.352	0.326
4	0.610	0.788	0.882
5	0.378	0.147	0.221
6	0.204	0.030	0.078
7	0.581	0.378	0.370
8	0.621	0.097	0.180
9	0.082	0.177	0.220
10	0.192	0.445	0.432
11	0.096	0.717	0.720
12	0.257	0.714	0.681
13	0.181	0.852	0.823
14	0.051	0.366	0.366
15	0.044	0.049	0.130
16	0.067	0.065	0.111
17	0.088	0.380	0.437
18	0.130	0.456	0.503
19	0.206	0.112	0.091

Example key phrase extraction:

- Endocrine, Nutritional, Metabolic

    - Blood sugar admiss, blood sugar also, blood sugar meal, cardiac diabet diet, cardiac diet, cardiac healthi diet, diet, diet abl, diet also, diet cardiac, diet consist, diet discharg, diet discharg instruct, diet electrolyt, diet exercis, diet follow, diet followup, diet followup instruct, diet monitor, diet per, diet physic, diet postop, diet seen, diet throughout, diet use, diet well, electrolyt nutrit birth, endocrinologist, follow diet, follow endocrinologist, food diet, healthi diet, heart healthi diet, hyperglycemia, hyperglycemia discharg, hyperglycemia like, hyperglycemia major, hypertriglyceridemia, hyperuricemia, lipid, normal healthi diet
- Parasitic, Infections
    - Also infect, appear infect, bacteri infect, bacteria patient, chronic infect, concern infect, concern infecti, concern wound infect, develop infect, discuss infecti, due infecti, experi fever, experi follow fever, fever infect, fever infecti, followup infecti, found infect, fungal infect, fungal rash, infect appear, infect chronic, infect clinic, infect concern, infect due pneumophila, infect fever, infect patient, infect present, infect skin, infect wound, infect wound site, infecti, infectionsepsi, like infect, like infecti, may infecti, medic infect, pathogen, patient infect

Findings

- Discuss with respect to the hypothesis and results from the original paper From the precision, recall, and f1 stores calculated we conclude our first hypothesis to be confirmed as "KeyClass performs on par with the fully supervised baseline FasTag (Venkataramam et al.) on the MIMIC-III ICD-9 code assignment problem in terms of recall, precision, and f1 metrics."

Second, from a qualitative analysis of the outputted keywords and phrases from KeyClass we confirm "The KeyClass model is able to extract frequent keywords and phrases that are highly indicative of a particular class from the unlabeled text using a pre-trained language model"

Finally, in our ablation study we attempt to address a central claim of the paper "self-training improves downstream classifier performance" by running KeyClass without self-training. We found the model to do far worse by the metrics of precision, recall, and f1 without self-training, thus confirming this claim.

In [ ]:
```
# The model takes over an hour to run on our Colab environment event after drastically s
# As a result, the code and scripts are provided here for proof of effort, but we did no
# to display the results without just uploading a pre-run notebook from a local machine.
# We managed to get a subpar F1 score of 0.51, compared to the paper's higher F1 score o
```

Discussion

The original KeyClass paper used a multidimensional multi-hot output vector. In our initial experiments, we ran through several attempts and seemed to fail to reproduce this dimension of the paper. Our experiments seemed to be incredibly computationally intensive, even if we reduced the size of data dramatically or the number of dimensions from the entire set (19).

In order to produce some output, despite it may have diverging from the paper, we looked into alternative methods of multiclass classification. So, we implemented one vs rest or one vs all classification instead. In this method, we generated the N-binary classifier models, effectively reducing the task into several binary classification problems.

- “What was easy” The paper for the most part was straightforward in their claims and provided a good suite of code to use for our intial reproduction steps.

- “What was difficult” Being unable to reproduce the multi-hot encoding methodology led us to a major roadblock. We were not sure if our implementation was incorrect, or if we simply lacked computational power.

- Recommendations to the original authors or others who work in this area for improving reproducibility The demo given alongside the original code could have been made more robust, on a larger set to give a good idea how long the runtime would be realistically. Clarification to the implementation of the multidimensional classification would be helpful. Also, ensuring the current python library requirements are up to date and compatible.

In [ ]:
```
# Our future plans will incorporate the ablations mentioned earlier.
# It was exceedingly difficult to glue together all the pieces of KeyClass
# due to very poor documentation on the part of the researchers.
# In particular, there is no documentation on the formatting of the input,
# which must be derived from the code.
# While the paper's original performance might be reproducible with enough compute power
# we were not able to achieve the same numbers as the paper.
# We will also clean up our code in the next phase.
```

References

1. Song, K., Tan, X., Qin, T., Lu, J., & Liu, T. Y. (2020). Mpnet: Masked and permuted pre-training for language understanding. Advances in neural information processing systems, 33, 16857-16867.

2. Venkataraman, G. R., Pineda, A. L., Bear Don’t Walk IV, O. J., Zehnder, A. M., Ayyar, S., Page, R. L., ... & Rivas, M. A. (2020). FasTag: Automatic text classification of unstructured medical narratives. PLoS one, 15(6), e0234647.