CS 598 Deep Learning for Healthcare

Final Project Draft

Team 140: Anson Wong & Skyler MacAdam

GitHub link: https://github.com/skylernovak/KeyClass/tree/main

For our final project, we have selected the paper Classifying Unstructured Clinical Notes via Automatic Weak Supervision. We will attempt to reproduce their study with similar results.

Mount Notebook to Google Drive

This step needs to be completed the first time when Google Colab is opened and you are starting a new session.

ATTENTION TA:

If you have issues with the mounted drive please let us know so we can resolve the issue. We have had some issues ourselves, but according to the instructions we believe we are supposed to be using the mounted drive?

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Introduction

Background

Assigning accurate diagnostic codes from systems like the International Classification of Diseases (ICD) to clinical notes is a critical task for healthcare providers. These codes are used for clinical documentation, research, and billing purposes. However, manually coding unstructured clinical notes is a time-consuming, costly, and error-prone process. Estimates suggest that 60-80% of manually assigned codes may not accurately reflect the true patient diagnoses[1].

Problem Statement

The problem tackled in this paper is the automatic assignment of ICD-9 diagnostic codes to unstructured patient discharge summaries. This is a challenging multiclass, multilabel text classification task, as there are over 14,000 unique ICD-9 codes that may need to be applied to a single clinical note. The paper examines other multiclass test classification problems with other datasets to show the possabilities that KeyClass can be used for, and argues it can be used for high level ICD-9 code classification on clinical notes if it can perform successfully on these other problems too. One of the datasets used most in the papers discussion is the IMDB dataset, where KeyClass is identifying if a movie review is positive or negative. KeyClass is given some keywords that a domain expert would provide (or they can also be obtained from inferences from related sources such as Wikipedia), and then able to predict the label of the reviews itself.

Importance and Difficulty

Accurate ICD coding is important for several reasons:

- It allows healthcare providers to better manage patient populations and track disease prevalence.
- It is crucial for correct billing and reimbursement, with coding errors potentially leading to revenue loss or even penalties.
- It enables large-scale clinical research by facilitating the identification of relevant patient cohorts.

However, the task is difficult due to the complexity and ever-evolving nature of ICD coding systems, as well as the unstructured and ambiguous nature of clinical narratives.

Prior work has explored the use of supervised machine learning techniques to automate ICD code assignment, such as hierarchical attention models[2] and multimodal approaches[3]. While these methods have shown promising results, they rely on large amounts of manually annotated training data, which can be costly and time-consuming to obtain. Additionally, as the ICD coding system is periodically revised, these supervised models may struggle to generalize to new codes.

Paper Explanation

The paper *Classifying Unstructured Clinical Notes via Automatic Weak Supervision* proposes a novel approach called KeyClass that aims to address the limitations of existing supervised methods. KeyClass is a weakly supervised text classification framework that learns from class-label descriptions alone, without requiring any human-labeled documents.

The key innovations of KeyClass are:

- Automatically generating interpretable keyword-based labeling functions from class descriptions using pre-trained language models.
- Employing data programming to estimate probabilistic labels from the noisy, automatically generated labeling functions.
- Training a downstream text classifier in a self-supervised manner, using the probabilistic labels.

The paper reports that KeyClass achieves performance comparable to state-of-the-art weakly supervised methods based on 4 real world test classificatgion datasets.

Contributions

This work is significant as it presents a novel weakly supervised approach that can effectively solve the important problem of automated ICD coding, without the need for extensive manual labeling. By leveraging only class descriptions, KeyClass has the potential to make the development and deployment of accurate text classifiers more accessible and affordable, with broader implications for clinical text processing and healthcare data management.

Scope of Reproducibility:

We will be attempting to reproduce the following hypothesis from the KeyClass paper:

1. KeyClass can effectively classify text documents without access to any labeled data.
2. Self-training improves the performance of the downstream classifier.

Methodology

Here, we begin our implementation of our study. Below you will find two sections, Data and Model.

Data

This section we will load and pre-process our data for our experiements.

Model

This section we will construct the model used for the experiment.

```
# Run this cell to install all requirements for this project.
%cd /content/drive/My Drive/CS598FinalProject/
%ls
!pip install -r requirements.txt
```

```
/content/drive/My Drive/CS598FinalProject
config_files/                          end_model_preds_train_key_class.npy  scripts/
data/                                  keyclass/                            y_test.npy
DL4H_Team_140                          models/                              y_train_lm_masked.npy
end_model_preds_test_key_class.npy     requirements.txt                     y_train_masked.npy
end_model_preds_test_self_training.npy results/
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 1)) (2.2.1+
Requirement already satisfied: transformers==4.21.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 2))
Requirement already satisfied: sentence-transformers>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (
Requirement already satisfied: numpy>=1.19.2 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 4)) (1.25.
Requirement already satisfied: scipy>=1.5.2 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 5)) (1.11.4
Requirement already satisfied: scikit-learn>=0.24.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 6))
Requirement already satisfied: tqdm>=4.59.0 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 7)) (4.66.2
Requirement already satisfied: snorkel>=0.9.6 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 8)) (0.9.
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers==4.21.1->-r requirements.txt
Requirement already satisfied: huggingface-hub<1.0,>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from transformers==4.21.1->
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers==4.21.1->-r requireme
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers==4.21.1->-r requirements.
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers==4.21.1->-r require
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers==4.21.1->-r requirements.txt
Requirement already satisfied: tokenizers!=0.11.3,<0.13,>=0.11.1 in /usr/local/lib/python3.10/dist-packages (from transformers==4.
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requirem
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requirements.txt (line 1))
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requirements.txt (line 1
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requirements.txt (line 1))
```

```
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requirements.txt (line 1))
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requi
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requ
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requ
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r re
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requirem
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requir
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->-r requirements.txt (l
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.10/dist-packages (from nvidia-cusolver-cu12==11.4.5
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (from sentence-transformers>=2.2.0->-r requi
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (from sentence-transformers>=2.2.0->-r requirements
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.10/dist-packages (from sentence-transformers>=2.2.0->-r req
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.24.1->-r requirement
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.24.1->-r requ
Requirement already satisfied: munkres>=1.0.6 in /usr/local/lib/python3.10/dist-packages (from snorkel>=0.9.6->-r requirements.txt
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from snorkel>=0.9.6->-r requirements.txt
Requirement already satisfied: tensorboard>=2.9.1 in /usr/local/lib/python3.10/dist-packages (from snorkel>=0.9.6->-r requirements
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->snorkel>=0.9
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->snorkel>=0.9.6->-r req
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->snorkel>=0.9.6->-r r
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=2.9.1->snorkel>=0.9.6->-
Requirement already satisfied: grpcio>=1.48.2 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=2.9.1->snorkel>=0.9.6-
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=2.9.1->snorkel>
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=2.9.1->s
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard>=2.9.1->snorkel>=0.9.6
```

```
# Run this cell to import the remaining modules needed for this experiment.
%cd /content/drive/My Drive/CS598FinalProject/keyclass/
%ls

# Append paths for imports.
import sys
sys.path.append('/content/drive/My Drive/CS598FinalProject/keyclass/')
sys.path.append('/content/drive/My Drive/CS598FinalProject/scripts/')

# Import files and modules needed for KeyClass
import numpy as np
import utils
import models
import create_lfs
import train_classifier
import torch
import pickle
from os.path import join, exists
import pandas as pd
import os
import train_downstream_model
```

```
/content/drive/My Drive/CS598FinalProject/keyclass
create_lfs.py  __init__.py  models.py  __pycache__/  train_classifier.py  utils.py
['/content', '/env/python', '/usr/lib/python310.zip', '/usr/lib/python3.10', '/usr/lib/python3.10/lib-dynload', '', '/usr/local/lib/pyth
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Data

Source of the data

The data used in the paper is collected from 4 different datasets. Amazon, DBpedia, IMDB, and AG News. The authors collected this data and included a script with their paper to download copies of these datasets as well for others to reproduce the authors findings. We downloaded this data and included it within out google drive as well. These scripts are downloaded using the script `get_data.sh`.

These 4 datasets are used in common multiclass text classification problems, such as in movie review sentiments from IMBD.

Statistics

Utilizing the IMDB dataset, the paper attempts to classify movie reviews as either `positive` or `negative`. The KeyClass model was provided with common sense descriptions for these classes from industry experts. This constitutes minimal human input required for KeyClass to perform the classifications. For example, a positive review is typically assocaited with words such as "amazing", "exciting", or "fun" while negative reviews are more often associated with words such as "terrible", "boring", or "awful".

Both the `train` and `test` datasets contains 25000 sample movie reviews each. KeyClass attempts to label each review as either `positive` or `negative` based on these reviews. Each data split contains approximately 32 million characters, and the files are approximately 30 MB each.

Data process

The KeyClass framework consists of several key steps:

1. Finding relevant keywords and phrases from the class descriptions using a pre-trained language model (such as BERT).
2. Constructing labeling functions based on the identified keywords and using data programming to generate probabilistic labels for the training data.
3. Training a downstream classifier on the probabilistically labeled training data.
4. Self-training the downstream model on the entire training dataset to refine the classifier.

```
# Declare configuration variables
config_file_path = r'/content/drive/My Drive/CS598FinalProject/config_files/config_imdb.yml' # Specify path to the configuration file
random_seed = 0 # Random seed for experiments
args = utils.Parser(config_file_path=config_file_path).parse()
```

The following block of code processes the text and creates embeddings. It is highly encouraged to use the Google Colab t4 GPU runtime type to execute this block of code. Using the T4 GPU, this takes a couple of minutes to execute. Using the CPU runtime type, this can take hours.

This block of code has been commented out as the embeddings have been saved in pickle files for later use in this notebook. This step is not required to be executed again, and is left here for informational / demonstrative purposes.

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

# if args['use_custom_encoder']:
#     model = models.CustomEncoder(pretrained_model_name_or_path=args['base_encoder'],
#         device='cuda' if torch.cuda.is_available() else 'cpu')
# else:
#     model = models.Encoder(model_name=args['base_encoder'],
#         device='cuda' if torch.cuda.is_available() else 'cpu')

# for split in ['train', 'test']:
#     sentences = utils.fetch_data(dataset=args['dataset'], split=split, path=args['data_path'])
#     embeddings = model.encode(sentences=sentences, batch_size=args['end_model_batch_size'],
#                               show_progress_bar=args['show_progress_bar'],
#                               normalize_embeddings=args['normalize_embeddings'])
#     with open(join(args['data_path'], args['dataset'], f'{split}_embeddings.pkl'), 'wb') as f:
#         pickle.dump(embeddings, f)
```

Here we now load the training data, and then create the labeling function. Finally we create the probablistic labels from the training document.

```
# Load training data
train_text = utils.fetch_data(dataset=args['dataset'], path=args['data_path'], split='train')

training_labels_present = False
if exists(join(args['data_path'], args['dataset'], 'train_labels.txt')):
    with open(join(args['data_path'], args['dataset'], 'train_labels.txt'), 'r') as f:
        y_train = f.readlines()
    y_train = np.array([int(i.replace('\n','')) for i in y_train])
    training_labels_present = True
else:
    y_train = None
    training_labels_present = False
    print('No training labels found!')

with open(join(args['data_path'], args['dataset'], 'train_embeddings.pkl'), 'rb') as f:
    X_train = pickle.load(f)

# Print dataset statistics
print(f"Getting labels for the {args['dataset']} data...")
print(f'Size of the data: {len(train_text)}')
if training_labels_present:
    print('Class distribution', np.unique(y_train, return_counts=True))

# Load label names/descriptions
label_names = []
for a in args:
    if 'target' in a: label_names.append(args[a])

# Creating labeling functions
labeler = create_lfs.CreateLabellingFunctions(base_encoder=args['base_encoder'],
                                              device=torch.device(args['device']),
                                              label_model=args['label_model'])
proba_preds = labeler.get_labels(text_corpus=train_text, label_names=label_names, min_df=args['min_df'],
                                 ngram_range=args['ngram_range'], topk=args['topk'], y_train=y_train,
                                 label_model_lr=args['label_model_lr'], label_model_n_epochs=args['label_model_n_epochs'],
                                 verbose=True, n_classes=args['n_classes'])

y_train_pred = np.argmax(proba_preds, axis=1)

# Save the predictions
if not os.path.exists(args['preds_path']): os.makedirs(args['preds_path'])
with open(join(args['preds_path'], f"{args['label_model']}_proba_preds.pkl"), 'wb') as f:
    pickle.dump(proba_preds, f)

# Print statistics
print('Label Model Predictions: Unique value and counts', np.unique(y_train_pred, return_counts=True))
if training_labels_present:
    print('Label Model Training Accuracy', np.mean(y_train_pred==y_train))

    # Log the metrics
    training_metrics_with_gt = utils.compute_metrics(y_preds=y_train_pred, y_true=y_train, average=args['average'])
    utils.log(metrics=training_metrics_with_gt, filename='label_model_with_ground_truth',
        results_dir=args['results_path'], split='train')
```

```
Getting labels for the imdb data...
Size of the data: 25000
Class distribution (array([0, 1]), array([12500, 12500]))
```

.gitattributes: 100% 744/744 [00:00<00:00, 8.93kB/s]

1_Pooling/config.json: 100% 190/190 [00:00<00:00, 8.33kB/s]

README.md: 100% 3.73k/3.73k [00:00<00:00, 59.7kB/s]

config.json: 100% 594/594 [00:00<00:00, 10.2kB/s]

config_sentence_transformers.json: 100% 122/122 [00:00<00:00, 8.75kB/s]

model.safetensors: 100% 438M/438M [00:02<00:00, 210MB/s]

pytorch_model.bin: 100% 438M/438M [00:02<00:00, 215MB/s]

sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 3.65kB/s]

special_tokens_map.json: 100% 239/239 [00:00<00:00, 16.0kB/s]

tokenizer.json: 100% 466k/466k [00:00<00:00, 9.65MB/s]

tokenizer_config.json: 100% 1.19k/1.19k [00:00<00:00, 92.0kB/s]

vocab.txt: 100% 232k/232k [00:00<00:00, 11.0MB/s]

modules.json: 100% 229/229 [00:00<00:00, 17.5kB/s]

```
Found assigned category counts [6789 9578]
labeler.vocabulary:
 16367
labeler.word_indicator_matrix.shape (25000, 600)
Len keywords 600
assigned_category: Unique and Counts (array([0, 1]), array([300, 300]))
negative, hate, expensive, bad, poor, broke, waste, horrible, would not recommend ['abominable' 'abomination' 'absolute worst' 'absolute
 'absolutely terrible' 'abuse' 'abused' 'abusive' 'abysmal'
 'acting horrible' 'acting poor' 'acting terrible' 'actors bad'
 'actually bad' 'also bad' 'among worst' 'annoyance' 'annoying' 'appalled'
 'appalling' 'atrocious' 'awful' 'awfully' 'awfulness' 'bad' 'bad actor'
 'bad actors' 'bad actually' 'bad almost' 'bad bad' 'bad could'
 'bad either' 'bad enough' 'bad even' 'bad film' 'bad films' 'bad get'
 'bad horror' 'bad idea' 'bad like' 'bad made' 'bad makes' 'bad many'
 'bad movie' 'bad movies' 'bad music' 'bad one' 'bad ones' 'bad people'
 'bad really' 'bad reviews' 'bad special' 'bad story' 'bad taste'
 'bad thing' 'bad things' 'bad think' 'bad way' 'bad well' 'bad would'
 'baddies' 'badly' 'badly made' 'badness' 'best worst' 'better worse'
 'big disappointment' 'chose' 'complain' 'complained' 'complaining'
 'complains' 'crap like' 'crappy' 'criticism' 'criticisms' 'criticize'
 'criticized' 'cursed' 'cursing' 'cynical' 'cynicism' 'depressed'
 'depressing' 'despair' 'desperation' 'despicable' 'despise'
 'disappointed' 'disappointing' 'disappointment' 'disastrous' 'disdain'
 'disgust' 'disgusted' 'disgusting' 'dislike' 'disliked' 'dismal'
 'displeasure' 'distasteful' 'distraught' 'dreaded' 'dreadful'
 'dreadfully' 'easily worst' 'else say' 'even bad' 'even worst'
 'far worse' 'far worst' 'feel bad' 'feel sorry' 'film awful' 'film bad'
 'film terrible' 'film worst' 'films bad' 'filth' 'filthy' 'get bad'
 'god awful' 'greed' 'hapless' 'hate' 'hate film' 'hate movie' 'hated'
 'hated movie' 'hateful' 'hates' 'hating' 'hatred' 'hideous' 'hideously'
 'honestly say' 'horrendous' 'horrible' 'horrible film' 'horrible movie'
 'horribly' 'horrid' 'horrific' 'huge disappointment' 'humble opinion'
 'idiotic' 'immoral' 'incredibly bad' 'incredibly stupid' 'irresponsible'
 'know bad' 'lack' 'lackluster' 'laughably bad' 'least good' 'like bad'
 'like cheap' 'like horror' 'like least' 'like say' 'loathing' 'lot bad'
 'lousy' 'love bad' 'love hate' 'low quality' 'made bad'
 'major disappointment' 'make bad' 'many bad' 'mean spirited' 'mediocrity'
 'miserable' 'miserably' 'misery' 'misfortune' 'misguided' 'movie awful'
 'movie bad' 'movie horrible' 'movie terrible' 'movie worst' 'movies bad'
 'much worse' 'nasty' 'needless say' 'needlessly' 'negative comments'
 'negative reviews' 'never want' 'nothing good' 'notorious' 'one awful'
 'one bad' 'one worst' 'opinion' 'opinions' 'pathetic' 'people hate'
 'pity' 'plain awful' 'plain bad' 'plain stupid' 'poor' 'poor quality'
 'poorest' 'poorly' 'possibly worst' 'poverty' 'pretty awful' 'pretty bad'
 'pretty lame' 'pretty poor' 'probably worst' 'quite bad' 'rather poor'
 'really annoying' 'really awful' 'really bad' 'really disappointed'
 'really hate' 'really poor' 'really sad' 'really stupid'
 'really terrible' 'refuse' 'regret' 'regrets' 'reject' 'repulsive'
 'rotten' 'sad' 'sad thing' 'saddest' 'sadistic' 'sadly' 'sadness'
 'say bad' 'say worst' 'see bad' 'seedy' 'simply awful' 'something bad'
 'still bad' 'story bad' 'stupid' 'stupidest' 'stupidity' 'stupidly'
 'sucks' 'terrible' 'terrible film' 'terrible movie' 'terribly'
 'think bad' 'tiresome' 'trash' 'trashy' 'truly awful' 'truly bad' 'ugly'
 'unappealing' 'unattractive' 'unbearably' 'uneducated' 'unfortunate'
 'unfortunately' 'unfortunately film' 'unhappy' 'unimaginative'
 'unimpressive' 'uninteresting' 'unlikable' 'unlikeable' 'unlucky'
 'unnecessarily' 'unpleasant' 'unrealistic' 'unremarkable' 'unsatisfied'
```

```
 'unsatisfying' 'unsympathetic' 'unwilling' 'waste money' 'worse'
 'worse movie' 'worse movies' 'worst' 'worst acting' 'worst ever'
 'worst film' 'worst films' 'worst kind' 'worst movie' 'worst movies'
 'worst part' 'worst thing' 'worthless' 'wretched' 'writing bad']
good, positive, excellent, amazing, love, fine, good quality, would recommend ['actually good' 'actually like' 'admirable' 'admirably' '
 'almost good' 'also enjoyed' 'also excellent' 'also good' 'also great'
 'also interesting' 'also like' 'also liked' 'also nice' 'also pretty'
 'also well' 'always good' 'always great' 'among best' 'another good'
 'another great' 'another reviewer' 'anyone likes' 'anything good'
 'award best' 'bad good' 'best' 'best best' 'best ever' 'best one'
 'best parts' 'best performance' 'best show' 'best thing' 'best things'
 'better ones' 'certainly good' 'commendable' 'damn good'
 'definitely good' 'definitely recommend' 'definitely worth' 'done good'
 'done great' 'enjoy' 'enjoy good' 'enjoyable' 'enjoyable film'
 'enjoyable movie' 'enjoyable watch' 'enough good' 'entertainment value'
 'especially good' 'especially like' 'especially liked' 'even good'
 'even great' 'excellence' 'excellent' 'excellent film' 'excellent job'
 'excellent movie' 'excellent performance' 'excellent performances'
 'excellently' 'exquisite' 'extremely well' 'fabulous' 'fairly good'
 'fantastic' 'far best' 'film excellent' 'find good' 'fine job'
 'fine performances' 'finest' 'first rate' 'get good' 'give good'
 'gives best' 'gives good' 'gives great' 'good' 'good action' 'good also'
 'good although' 'good bad' 'good choice' 'good direction' 'good either'
 'good enough' 'good entertainment' 'good especially' 'good even'
 'good example' 'good film' 'good films' 'good first' 'good good'
 'good great' 'good idea' 'good movie' 'good music' 'good one' 'good ones'
 'good original' 'good part' 'good parts' 'good people' 'good performance'
 'good performances' 'good really' 'good reviews' 'good say' 'good show'
 'good special' 'good stuff' 'good taste' 'good thing' 'good things'
 'good think' 'good though' 'good tv' 'good use' 'good well' 'good work'
 'good would' 'good writing' 'got good' 'got great' 'great'
 'great character' 'great example' 'great film' 'great fun' 'great love'
 'great music' 'great one' 'great really' 'great show' 'great supporting'
 'great things' 'great time' 'greats' 'high quality' 'high rating'
 'highly recommend' 'highly recommended' 'however like' 'idea good'
 'like best' 'like good' 'like great' 'liked' 'liked one' 'looks great'
 'lot good' 'lot great' 'love good' 'lovely' 'luxury' 'made good'
 'made great' 'made well' 'make good' 'make great' 'makes good'
 'makes great' 'many good' 'many great' 'many reviewers' 'many reviews'
 'marvelously' 'may good' 'mean good' 'might good' 'movie excellent'
 'movie good' 'movie recommend' 'movie wonderful' 'much enjoyed'
 'much good' 'music good' 'music great' 'nearly good' 'nice' 'nice look'
 'nothing better' 'one best' 'one finest' 'one good' 'one great'
 'one like' 'overall good' 'overall think' 'particularly good'
 'people good' 'people like' 'performances good' 'perhaps best'
 'personal favorite' 'personally think' 'pleasant' 'positive reviews'
 'positive thing' 'possibly best' 'praise' 'prefer' 'prefers'
 'pretty decent' 'pretty good' 'probably best' 'probably good'
 'probably like' 'put good' 'qualities' 'quality' 'quality acting'
 'quite enjoyable' 'quite good' 'quite like' 'rather good' 'rating 10'
 'read review' 'read reviews' 'reading reviews' 'real good'
 'really appreciate' 'really enjoy' 'really enjoyed' 'really good'
 'really great' 'really like' 'really liked' 'really loved' 'really nice'
 'really recommend' 'recommend' 'recommend anyone' 'recommend everyone'
 'recommend film' 'recommend movie' 'recommend one' 'recommend see'
 'recommend watch' 'recommend watching' 'recommendation' 'recommended'
 'recommending' 'redeeming quality' 'reviews' 'satisfactory' 'say best'
 'say good' 'see good' 'seen good' 'show good' 'solid performances'
 'something better' 'something good' 'something interesting' 'splendid'
 'still enjoyable' 'still good' 'still great' 'strongly recommend'
 'surprisingly good' 'tasteful' 'terrific' 'thing good' 'think best'
 'think good' 'think great' 'though good' 'thought good' 'thought great'
 'time great' 'top notch' 'truly great' 'two best' 'want good'
 'watch good' 'well crafted' 'well good' 'well great' 'well made'
 'well produced' 'well worth' 'wonderful' 'wonderful film' 'wonderful job'
 'wonderful life' 'wonderful movie' 'wonderfully' 'worth look'
 'worth mentioning' 'worth seeing' 'worthwhile' 'would good'
 'would recommend']
==== Training the label model ====
100%|██████████| 100/100 [00:08<00:00, 11.32epoch/s]
Label Model Predictions: Unique value and counts (array([0, 1]), array([ 8914, 16086]))
Label Model Training Accuracy 0.70016
Saving results in ../results/imdb/train_label_model_with_ground_truth_14-Apr-2024-16_32_31.txt...
```

Model

The model includes the model definitation which usually is a class, model training, and other necessary parts.

- Model architecture: layer number/size/type, activation function, etc

- Training objectives: loss function, optimizer, weight of each loss term, etc
- Others: whether the model is pretrained, Monte Carlo simulation for uncertainty analysis, etc
- The code of model should have classes of the model, functions of model training, model validation, etc.
- If your model training is done outside of this notebook, please upload the trained model here and develop a function to load and test it.

Train the Downstream Model

Find Class Descriptions: KeyClass starts with just the class descriptions (e.g., "positive review" and "negative review") without any labeled training data. Find Relevant Keywords: Using the class descriptions and a pre-trained language model (like BERT), KeyClass automatically extracts keywords and phrases that are highly indicative of each class. Probabilistically Label the Data: KeyClass uses the extracted keywords as labeling functions and applies data programming techniques to generate probabilistic labels for the entire training dataset. Train Downstream Model: KeyClass then trains a downstream text classification model (e.g., a feed-forward neural network) using the probabilistically labeled training data. It initially only uses the most confidently labeled samples to train the model.

The key advantages of this approach are that it requires no manually labeled training data and the labeling functions are automatically generated in an interpretable way.

disabled for now

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

# # Set random seeds
# 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(args)

# # Train a downstream classifier

# if args['use_custom_encoder']:
#   encoder = models.CustomEncoder(pretrained_model_name_or_path=args['base_encoder'], device=args['device'])
# else:
#   encoder = models.Encoder(model_name=args['base_encoder'], device=args['device'])

# classifier = models.FeedForwardFlexible(encoder_model=encoder,
#                                         h_sizes=args['h_sizes'],
#                                         activation=eval(args['activation']),
#                                         device=torch.device(args['device']))
# print('\n===== Training the downstream classifier =====\n')
# model = train_classifier.train(model=classifier,
#                            device=torch.device(args['device']),
#                            X_train=X_train_embed_masked,
#                            y_train=y_train_lm_masked,
#                            sample_weights=sample_weights_masked if args['use_noise_aware_loss'] else None,
#                            epochs=args['end_model_epochs'],
#                            batch_size=args['end_model_batch_size'],
#                            criterion=eval(args['criterion']),
#                            raw_text=False,
#                            lr=eval(args['end_model_lr']),
#                            weight_decay=eval(args['end_model_weight_decay']),
#                            patience=args['end_model_patience'])


# end_model_preds_train = model.predict_proba(torch.from_numpy(X_train_embed_masked), batch_size=512, raw_text=False)
# end_model_preds_test = model.predict_proba(torch.from_numpy(X_test_embed), batch_size=512, raw_text=False)
```

Self-Train the Model

Encode Full Training Set: After the initial downstream model is trained, KeyClass encodes the full training dataset using the model's encoder. Self-Train the Model: KeyClass then iteratively self-trains the downstream model using the entire encoded training set. It does this by: Making predictions on the unlabeled training data Selecting the most confidently predicted samples Retraining the model on the combined labeled and pseudo-labeled data Evaluate on Test Set: The self-trained model is then evaluated on the held-out test set to measure the final performance.

The self-training process allows the model to refine its representations and decision boundaries by iteratively learning from its own predictions on the unlabeled data. This can lead to improved performance, especially when the initial model is trained on a limited subset of the data.

disabled for now

```
# # Fetching the raw text data for self-training
# X_train_text = utils.fetch_data(dataset=args['dataset'], path=args['data_path'], split='train')
# X_test_text = utils.fetch_data(dataset=args['dataset'], path=args['data_path'], split='test')

# model = train_classifier.self_train(model=model,
#                                     X_train=X_train_text,
#                                     X_val=X_test_text,
#                                     y_val=y_test,
#                                     device=torch.device(args['device']),
#                                     lr=eval(args['self_train_lr']),
#                                     weight_decay=eval(args['self_train_weight_decay']),
#                                     patience=args['self_train_patience'],
#                                     batch_size=args['self_train_batch_size'],
#                                     q_update_interval=args['q_update_interval'],
#                                     self_train_thresh=eval(args['self_train_thresh']),
#                                     print_eval=True)


# end_model_preds_test = model.predict_proba(X_test_text, batch_size=args['self_train_batch_size'], raw_text=True)


# # Print statistics
# testing_metrics = utils.compute_metrics_bootstrap(y_preds=np.argmax(end_model_preds_test, axis=1),
#                                                   y_true=y_test,
#                                                   average=args['average'],
#                                                   n_bootstrap=args['n_bootstrap'],
#                                                   n_jobs=args['n_jobs'])
# print(testing_metrics)
```

After training the downstream classifier and self-training it, the notebook performs the following evaluation steps:

Load Test Data: It loads the test dataset, including the test text samples (X_test_embed) and the ground truth test labels (y_test). Evaluate Trained Model on Test Set: It uses the trained downstream model to make predictions on the test set, obtaining the test set predictions (end_model_preds_test). Compute Test Metrics: It computes various performance metrics on the test set predictions, including:

Metrics using ground truth labels (y_test): Compute metrics using utils.compute_metrics_bootstrap(), which does bootstrap sampling to get confidence intervals. This provides an assessment of the model's true performance on the test set.

Metrics using label model predictions (y_train_lm_masked): Compute metrics using utils.compute_metrics(). This shows how the model performs compared to the noisy labels used for training.

Print Test Metrics: The notebook prints out the test set performance metrics, showing the model's accuracy, precision, recall, and F1-score. Self-Train the Model: Finally, it loads the self-trained model checkpoint and evaluates the self-trained model on the test set, printing the updated test set performance metrics.

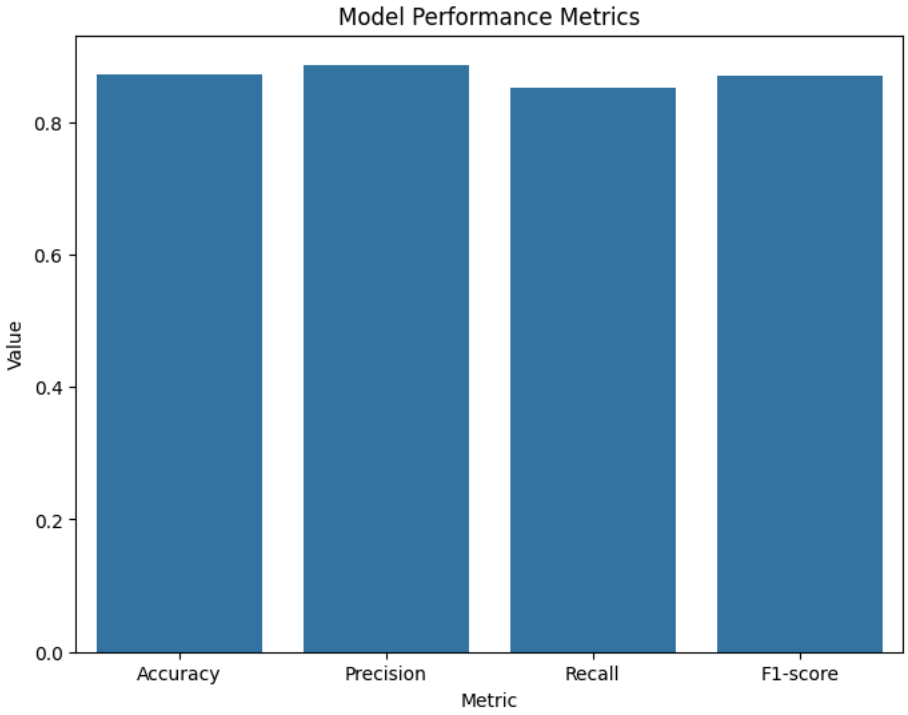
```
end_model_path='/content/drive/My Drive/CS598FinalProject/models/end_model.pth'
end_model_self_trained_path='/content/drive/My Drive/CS598FinalProject/models/end_model_self_trained.pth'

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

# Set random seeds
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(args)

model = torch.load(end_model_path)

end_model_preds_train_key_class = model.predict_proba(torch.from_numpy(X_train_embed_masked), batch_size=512, raw_text=False)
end_model_preds_test_key_class = model.predict_proba(torch.from_numpy(X_test_embed), batch_size=512, raw_text=False)

# Print statistics
if training_labels_present:
    training_metrics_with_gt = utils.compute_metrics(y_preds=np.argmax(end_model_preds_train_key_class, axis=1),
                                                     y_true=y_train_masked,
                                                     average=args['average'])
    print('training_metrics_with_gt', training_metrics_with_gt)

training_metrics_with_lm = utils.compute_metrics(y_preds=np.argmax(end_model_preds_train_key_class, axis=1),
                                                 y_true=y_train_lm_masked,
                                                 average=args['average'])
print('training_metrics_with_lm', training_metrics_with_lm)

testing_metrics = utils.compute_metrics_bootstrap(y_preds=np.argmax(end_model_preds_test_key_class, axis=1),
                                                  y_true=y_test,
                                                  average=args['average'],
                                                  n_bootstrap=args['n_bootstrap'],
                                                  n_jobs=args['n_jobs'])
print('testing_metrics', testing_metrics)


print('\n===== Self-training the downstream classifier =====\n')

# Fetching the raw text data for self-training
X_train_text = utils.fetch_data(dataset=args['dataset'], path=args['data_path'], split='train')
X_test_text = utils.fetch_data(dataset=args['dataset'], path=args['data_path'], split='test')

model = torch.load(end_model_self_trained_path)

end_model_preds_test_self_training = model.predict_proba(X_test_text, batch_size=args['self_train_batch_size'], raw_text=True)


# Print statistics
testing_metrics = utils.compute_metrics_bootstrap(y_preds=np.argmax(end_model_preds_test_self_training, axis=1),
                                                  y_true=y_test,
                                                  average=args['average'],
                                                  n_bootstrap=args['n_bootstrap'],
                                                  n_jobs=args['n_jobs'])
print('testing_metrics after self train', testing_metrics)
```

```
Confidence of least confident data point of class 0: 0.9118951704039087
Confidence of least confident data point of class 1: 0.9999157389338196

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

KeyClass only trains on the most confidently labeled data points! Applying mask...

==== Data statistics (after applying mask) ====
Size of training data: (7000, 768)
Size of training labels: (7000,)
Training class distribution (ground truth): [0.55057143 0.44942857]
Training class distribution (label model predictions): [0.5 0.5]
training_metrics_with_gt [0.9211428571428572, 0.9243499652296476, 0.9211428571428572]
training_metrics_with_lm [0.9188571428571428, 0.9190760887320724, 0.9188571428571428]
[Parallel(n_jobs=10)]: Using backend LokyBackend with 10 concurrent workers.
[Parallel(n_jobs=10)]: Done  30 tasks      | elapsed:   11.0s
```

```
[Parallel(n_jobs=10)]: Done  81 out of 100 | elapsed:   11.8s remaining:    2.8s
[Parallel(n_jobs=10)]: Done 100 out of 100 | elapsed:   20.1s finished
testing_metrics [[0.8533216  0.00209534]
 [0.86366085 0.00188531]
 [0.8533216  0.00209534]]

===== Self-training the downstream classifier =====

[Parallel(n_jobs=10)]: Using backend LokyBackend with 10 concurrent workers.
[Parallel(n_jobs=10)]: Done  40 tasks      | elapsed:    0.7s
testing_metrics after self train [[0.8714468  0.00204239]
 [0.87203099 0.0020189 ]
 [0.8714468  0.00204239]]
[Parallel(n_jobs=10)]: Done 100 out of 100 | elapsed:    1.4s finished
```

Results

```
file_path = '/content/drive/My Drive/CS598FinalProject/y_test.npy'
y_test = np.load(file_path)


file_path = '/content/drive/My Drive/CS598FinalProject/y_train_masked.npy'
y_train_masked = np.load(file_path)


file_path = '/content/drive/My Drive/CS598FinalProject/y_train_lm_masked.npy'
y_train_lm_masked = np.load(file_path)


file_path = '/content/drive/My Drive/CS598FinalProject/end_model_preds_train_key_class.npy'
end_model_preds_train_key_class = np.load(file_path)


file_path = '/content/drive/My Drive/CS598FinalProject/end_model_preds_test_key_class.npy'
end_model_preds_test_key_class = np.load(file_path)


file_path = '/content/drive/My Drive/CS598FinalProject/end_model_preds_test_self_training.npy'
end_model_preds_test_self_training = np.load(file_path)


import matplotlib.pyplot as plt
import seaborn as sns


# weak supervision sources (keywords and phrases) results


from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Compute the metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Create a bar plot
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-score']
values = [accuracy, precision, recall, f1]
plt.figure(figsize=(8, 6))
sns.barplot(x=metrics, y=values)
plt.title('Model Performance Metrics')
plt.xlabel('Metric')
plt.ylabel('Value')
plt.show()
```


Model Performance Metrics
Value
0.0
0.2
0.4
0.6
0.8
Accuracy
Precision
Recall
F1-score
Metric

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_curve, auc
import seaborn as sns

# Plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title='Confusion Matrix', labels=None):
    cm = confusion_matrix(y_true, y_pred)
    if labels:
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
    else:
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(title)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

# Plot precision-recall curve
def plot_precision_recall_curve(y_true, y_score, title):
    precision, recall, _ = precision_recall_curve(y_true, y_score)
    plt.plot(recall, precision, marker='.')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(title)
    plt.show()

# Plot ROC curve
def plot_roc_curve(y_true, y_score, title):
    fpr, tpr, _ = roc_curve(y_true, y_score)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc='lower right')
    plt.show()


# Plot histogram of predicted probabilities
def plot_predicted_probabilities_histogram(y_prob):
    plt.hist(y_prob, bins=10)
    plt.xlabel('Predicted Probability')
    plt.ylabel('Frequency')
    plt.title('Histogram of Predicted Probabilities')
    plt.show()


# Example usage
y_preds=np.argmax(end_model_preds_train_key_class, axis=1)

# plot_predicted_probabilities_histogram(end_model_preds_train_key_class)
# plot_probabilistic_vs_predicted_labels(y_train_lm_masked, y_preds)
```

The following evaluates how well the model learns from ground truth labels (y_train_masked) and the label model predictions (y_train_lm_masked)

Confusion Matrix:

The confusion matrix showcases how accurately the model predicts the true labels compared to the ground truth. It highlights the model's performance in terms of correctly identifying positive and negative instances and any misclassifications.

Precision-Recall Curve:

The precision-recall curve demonstrates the trade-off between precision and recall for different decision thresholds. It quantifies how well the model identifies positive instances while minimizing false positives and negatives compared to the ground truth.

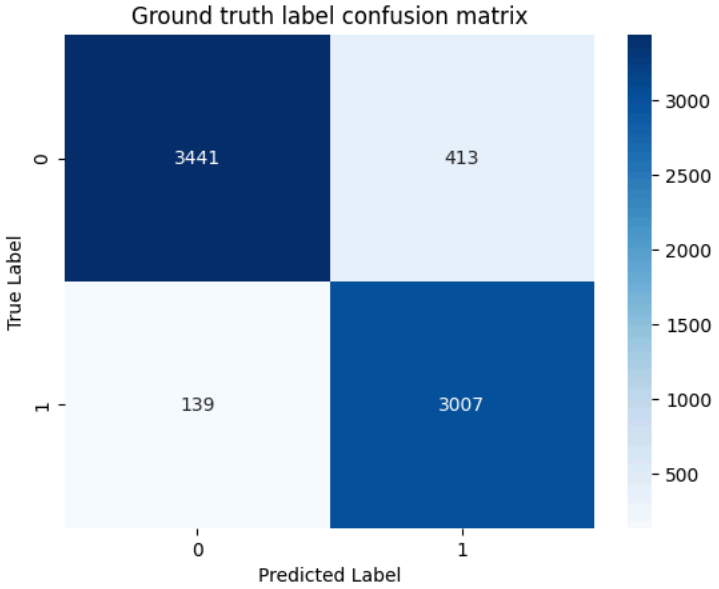
ROC Curve:

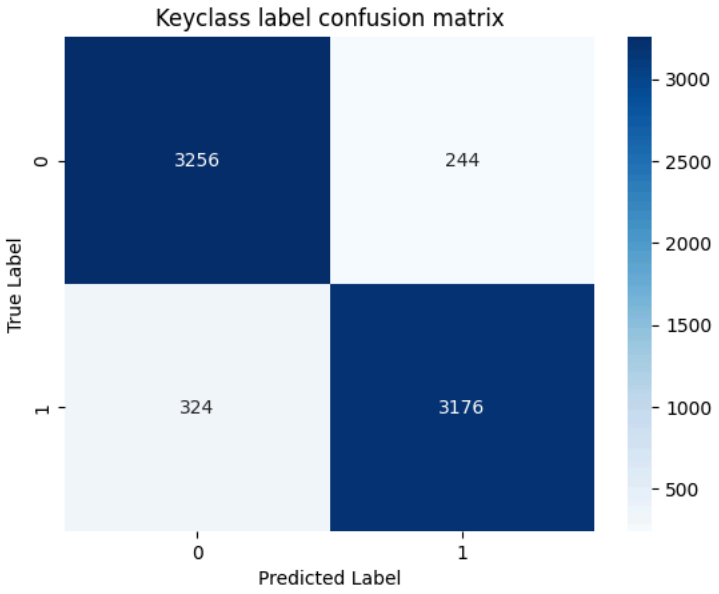
The ROC curve illustrates the model's ability to distinguish between positive and negative instances across various decision thresholds. It measures the model's true positive rate and false positive rate compared to the ground truth.

Overall, these comparisons of the plots between Ground Truth vs. Predicted Labels and KeyClass Labels vs. Predicted Labels are similar. It showcased the high performance of keyclass low supervised labeling.

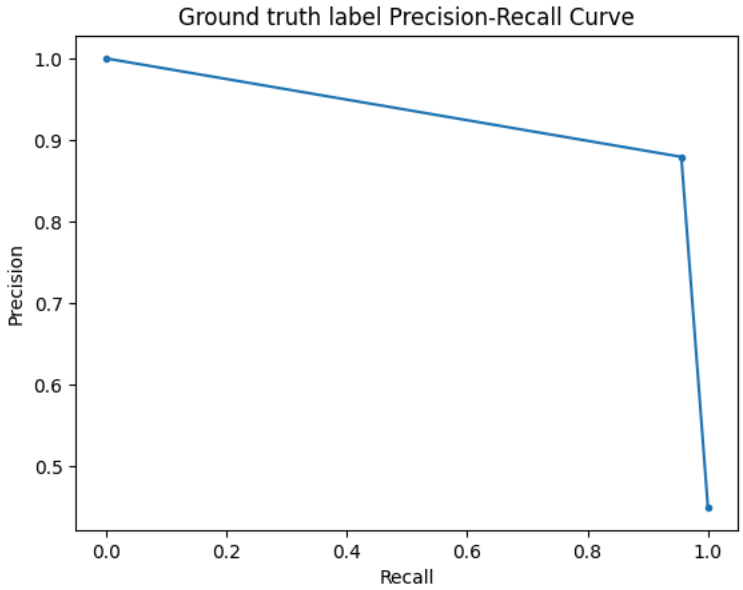
Confusion Matrix reveal how well the model is able to learn from the keyclass probabilistic labels compared to the true labels.

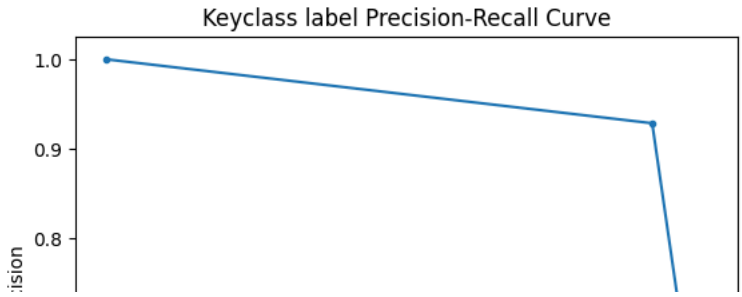
```
plot_confusion_matrix(y_train_masked, y_preds, 'Ground truth label confusion matrix')
plot_confusion_matrix(y_train_lm_masked, y_preds, 'Keyclass label confusion matrix')
```






```
plot_precision_recall_curve(y_train_masked, y_preds, 'Ground truth label Precision-Recall Curve')
plot_precision_recall_curve(y_train_lm_masked, y_preds, 'Keyclass label Precision-Recall Curve')
```






```
plot_roc_curve(y_train_masked, y_preds, 'Ground truth label ROC Curve')
plot_roc_curve(y_train_lm_masked, y_preds, 'Keyclass label ROC Curve')
```