CSI5386: Natural Language Processing
ASSIGNMENT 2: Text classification using deep learning

PROFESSOR: Dr. Diana Inkpen

Team : GROUP 27

TEAM MEMBERS	STUDENT ID	STUDENT EMAIL
Divya Reddy	300290332	dredd037@uottawa.ca
Ishveen Manjeet Singh Sahi	300303960	isahi065@uottawa.ca
Syeda Zainab	300303434	szain039@uottawa.ca

Task Distribution among team members:

Part 1 was completed by all three of us. Because it took a long time to execute and get the results.

Part 2 was trained by Syeda and pearson coefficient was calculated by Divya and Ishveen

Tasks done by Divya:
- Implemented the Part 1 of the code
- Ran the code of part 2 to get the pearson coefficient
- Contributed to the report for both Part1 and 2

Tasks done by Syeda:
- Implemented the Part 1 of the code
- Trained Sbert model
- Contributed to the report for both Part1 and 2

Tasks done by Ishveen:
- Implemented the Part 1 of the code
- Ran the code of part 2 to get the pearson coefficient
- Contributed to the report for both Part1 and 2

Part 1: Legal text classification

Programming Language used: Python

OBJECTIVE:

The objective is to extract and categorise the relevant text into the 41 classes while highlighting the key passages in a contract that are crucial for a human to review.

PROCEDURE:

The programming language used for this task is Python. The assignment was completed on the terminal on mac and Google Collab platform using the HuggingFace Transformers library, particularly the QuestionAnswering models included in the Transformers library.

DATASET USED:

For this assignment, we have used the Contract Understanding Atticus Dataset (CUAD) v1. The Atticus Project's CUAD v1 dataset is a collection of roughly 13101 labels in 510 commercial legal contracts that were manually categorised to identify 41 kinds of crucial provisions that lawyers look for when assessing contracts.

The contracts were received from the Electronic Data Gathering, Analysis, and Retrieval (EDGAR) system of the U.S. Securities and Exchange Commission.
The files included in CUAD v1 are 1 CSV file, 1 SQuAD-style JSON file, 28 Excel files, 510 PDF files, and 510 TXT files.

PROCESS:

We used 3 trained and 3 pre-trained models on the given dataset. Pre-training of these models was done on the CUAD v1 dataset. The dataset is highly imbalanced between the balanced and unbalanced labels. The dataset comprises imbalanced data owing to numerous irrelevant sentences. Therefore, the evaluation metrics opted for is Precision, Recall and Area Under the Precision Recall Curve (AUPR).

Models used for training and prediction:

- For the given problem structure, we are using six different models 3 of which are pre trained and the other 3 were trained by us:

Pre-Trained Models:

1. **RoBERTa-base**
    - Robustly Optimized BERT Pre Training Approach (RoBERTa) base model is a pre-trained model. It was modeled on a corpus of English language utilising a masked language modeling (MLM) objective with no human involvement in labelling the texts. RoBERTa uses BERT-base

architecture.It comprises 12 layers, 768 hidden layers, 12 heads and 125M parameters. It is a case-sensitive model. This model is capable of learning a bi-directional representation of the sentence.

2. **RoBERTa-large**
   - RoBERTa uses BERT-large architecture.It was modeled on an extensive corpus of English language. It comprises 24 layers, 1024 hidden layers, 16 heads and 355M parameters. Like the RoBERTa-based model, this model is case-sensitive and is also capable of learning a bi-directional representation of the sentence.

3. **DeBERTa-xlarge**
   - Decoding-enhanced BERT with Disentangled Attention (DeBERTa) model was built to improve the performance of BERT and RoBERTa models. It uses two novel approaches. First, is the disentangled attention mechanism where every word is described by two vectors which encode the content and position of the word respectively. Weights for the words are computed using disentangled matrices using the two vectors. Second, a replacement to the output softmax layer is the enhanced mask decoder which is used to predict the masked tokens during the pre-training process.

IMPLEMENTATION:

1. Installed required transformers using pip install.
2. Created a copy of the existing git repo in our google colab notebook by using the git clone functionality.
3. Added the pre-trained models available on HuggingFace in google colab.
4. Executed the evaluation.py file to generate the results.

LANGUAGE MODEL NAME	AUPR	PRECISION AT 80% RECALL	PRECISION AT 90% RECALL
RoBERTa-base	0.4258562746289796	0.31130839761315676	0
RoBERTa-large	0.48249698739044944	0.38128096249115356	0
DeBERTa-xlarge	0.4779347388727093	0.4403155490969483	0.17828358208955225

The implementation of the three pre-trained models, RoBERTa- large is better in terms of AUPR and DeBERTa-xlarge is better in terms of precision at 80% recall.

RESULTS:
PreTrained Models:

1. **Model Name : roberta-base**
   Output:
   {
   'name': 'roberta-base',
   'aupr': 0.4258562746289796,
   'prec_at_80_recall': 0.31130839761315676,
   'prec_at_90_recall': 0
   }

```
!python /content/cuad/evaluate.py
```

```
AUPR: 0.426, Precision at 80% Recall: 0.311, Precision at 90% Recall: 0.000
```

2. **Model : roberta- large**
   Output:
   {
   'name': 'roberta-large',
   'aupr': 0.48249698739044944,
   'prec_at_80_recall': 0.38128096249115356,
   'prec_at_90_recall': 0
   }

```
Last login: Mon Nov 28 17:32:48 on ttys000
(base) divya@Divyas-MacBook-Pro ~ % cd code
(base) divya@Divyas-MacBook-Pro code % cd cuad-main
(base) divya@Divyas-MacBook-Pro cuad-main % python evaluate.py
AUPR: 0.482, Precision at 80% Recall: 0.381, Precision at 90% Recall: 0.000
```

3. **Model : DeBERTa-v2-xlarge**
   Output:
   {
   'name': 'DeBERTa-v2-xlarge',
   'aupr':0.4779347388727093,
   'prec_at_80_recall': 0.4403155490969483,
   'prec_at_90_recall': 0.17828358208955225
   }

```
(base) divya@Divyas-MacBook-Pro cuad-main % python evaluate.py
AUPR: 0.478, Precision at 80% Recall: 0.440, Precision at 90% Recall: 0.178
(base) divya@Divyas-MacBook-Pro cuad-main %
```

Models trained by us:

1. **bert-base-uncased-squad-v1**
    - The model bert base uncased-squad-v1 is a Natural Language Processing (NLP) Model that is implemented in the Transformer library, typically using the Python programming language. The model was adjusted from the HuggingFace BERT base uncased checkpoint on SQuAD1.1. The model is case-insensitive, meaning that it does not differentiate between English and English.
2. **bert-small-finetuned-cuad**
    - This model is a part of 24 smaller BERT models (English only, uncased, trained with WordPiece masking). The more compact BERT models are designed for settings with constrained computational resources. Similar to how the original BERT models were adjusted, these can also be adjusted.
3. **legal-bert-base-cuad**
    - The legal-bert-base-cuad model is a fine-tuned version of the legal-bert-base-uncased model on the CUAD dataset. The legal-bert model belongs to the group of BERT models having a legal domain which enables legal assistance to conduct various research pertaining to the NLP domain. In general, the legal-bert model performs better than BERT as is seen from our results as well.

LANGUAGE MODEL NAME	AUPR
bert-base-uncased-squad-v1	0.08757518950667581
bert-small-finetuned-cuad	0.15104571031973515
legal-bert-base-cuad	0.3036292229311512

IMPLEMENTATION:

1. Installed required transformers using pip install.
2. Created a copy of the existing git repo in our google colab notebook by using the git clone functionality.
3. Added the pre-trained models available on HuggingFace in google collab.
4. Created Corpus . Took 16 training files for training and 4 for testing.
5. Ran run.sh

6. Executed the evaluation.py file to generate the results.

Corpus creation:

The data was split into 80% for training and 20% for tests. We used 16 files for training the data and 4 files for testing.

```
import json

#read training data
with open("/content/cuad/data/train_separate_questions.json","r") as file:
  train_file = json.load(file)

#save the 16 file of training data in train_file
train_file["data"]=train_file["data"][0:16]
with open("/content/cuad/data/training_dataset.json","w") as file:
  json.dump(train_file, file)

#read the test file
with open("/content/cuad/data/test.json","r") as file:
  test_file=json.load(file)

#save 4 files in testing_dataset.json
test_file["data"]=test_file["data"][0:4]
with open("/content/cuad/data/testing_dataset.json","w") as file:
  json.dump(test_file,file)
```

Changes to parameters:

CUDA_VISIBLE_DEVICES=0,1 python /content/cuad/train.py \
--output_dir /content/train_models/roberta-base \
--model_type roberta \
--model_name_or_path /content/bert-base-uncased-squad-v1 \
--train_file /content/cuad/data/training_dataset.json \
--predict_file /content/cuad/data/testing_dataset.json \
--do_train \
--do_eval \
--version_2_with_negative \
--learning_rate 1e-4 \
--num_train_epochs 3 \
--per_gpu_eval_batch_size=4 \
--per_gpu_train_batch_size=4 \

--max_seq_length 512 \
--max_answer_length 512 \
--doc_stride 256 \
--save_steps 1000 \
--n_best_size 20 \
--overwrite_output_dir

We changed the epoch number to 3, eval_batch_size and train_batch size to 4 for faster execution. Directory and model path needs to be changed as per the model that needs to be trained.

RESULTS:
Trained Models :

1. **Model Name : bert-base-uncased-squad-v1**
   Output :
   {
   'name':'bert-base-uncased-squad-v1',
   'aupr': 0.08757518950667581,
   'prec_at_80_recall': 0, 'prec_at_90_recall': 0
   }

```
[11] !python /content/cuad/evaluate.py

     AUPR: 0.088, Precision at 80% Recall: 0.000, Precision at 90% Recall: 0.000
```

2. **Model Name : bert-small-finetuned-cuad**
   Output :
   {
   'name': 'bert-small-finetuned-cuad',
   'aupr': 0.15104571031973515,
   'prec_at_80_recall': 0,
   'prec_at_90_recall': 0
   }

```
!python /content/cuad/evaluate.py

AUPR: 0.151, Precision at 80% Recall: 0.000, Precision at 90% Recall: 0.000
```

3. **Model Name: legal-bert-base-cuad**
   Output:
   {
   'name': 'legal-bert-base-cuad',
   'aupr': 0.3036292229311512,
   'prec_at_80_recall': 0,
   'prec_at_90_recall': 0
   }

```
[6]  ! python /content/cuad/evaluate.py

     AUPR: 0.304, Precision at 80% Recall: 0.000, Precision at 90% Recall: 0.000
```

Amongst the 3 trained models (bert-base-uncased-squad-v1, bert-small-finetuned-cuad, legal-bert-base-cuad) legal-bert-base-cuad model yielded a higher AUPR score of 0.304 since it was fine-tuned.

Problems Faced:

We tried experimenting with numerous files but most of the time we were getting memory issues. After a lot of attempts and reducing the data size . 16:4(training:testing) ratio was executed. The code was run on google colab because our local systems were not able to support the load. With just 5 training files the train.py file ran for almost a day. 32:8 files ran for 3 hours and failed with a memory error.

Part 2: Training sentence similarity models

OBJECTIVE:

The goal is to use the same Semeval 2016-Task 1 Semantic Textual Similarity (STS) dataset as in Assignment 1 and to try to improve on that assignment's results by using any of the training data that was made available. Then, for the purpose of calculating sentence similarity, we used the best pre-trained model (Sentence Embedding) from Assignment 1, which in our case is SBERT.

DATASET DESCRIPTION:

Dataset for Training : Took STS 2012 training data and merged all the files in one file and all gs scores in another file. Downloaded it from the link below.

Training and Trial Data

- **STS Core** - English monolingual subtask:
  - All pairs released during prior STS evaluations are available as trial and training data.
    - **2012**: Trial (6), Train (2,234), Test (3,108)
    - **2013**: Test (1,500)
    - **2014**: Test (3,750)
    - **2015**: Test (3,000) Raw (8,500)
- **Cross-lingual STS** - English/Spanish subtask:
  - Trial data: Spanish-English STS Trial Pairs.

SBERT:

To improve the results from Assignment 1 we have chosen the pre-trained SBERT-MpNet sentence embedding model as it had the best results..

Sentence-BERT (SBERT) or Sentence-Bidirectional Encoder Representations from Transformers is a sentence embedding technique that fine-tunes models which are pre-trained in a Siamese network architecture on the Natural Language Inference (NLI) task. SBERT performs a pooling operation by sending each sentence of a pair to BERT and then obtains sentence embeddings from the output contextualized word embeddings.

The **SBERT- MpNet** model stands for Masked and Permuted Pre-training for Language Understanding. It adopts a novel pre-training method, named masked and permuted language modeling, to seek advantages of the masked and permuted language modeling for natural language understanding.

They have been extensively assessed for the quality of their sentence embeddings (Performance Sentence Embeddings) and for embedded search queries and paragraphs (Performance Semantic Search). This framework is best suited to compute sentence embeddings for various languages.

The resultant embeddings are efficient in finding sentences with similar meanings. The extent of similarity can be computed using Cosine-Similarity.

MODEL:

We have used SentenceTransformer to define the model. It is designed in a way that eases the process of fine-tuning the sentence embedding models.

We have defined the model using the 'all-mpnet-base-v2' pre-trained sentence-transformer model. It maps sentences to a 768 dimensional dense vector space. It can be effectively used to perform sentence similarity tasks.

```
#defining the model by loading a pre-trained model
model = SentenceTransformer('all-mpnet-base-v2')
model_save_path = 'model_all_mp_base_v2_e10_w1000_bs64/'
```

IMPLEMENTATION:

1. **Training**
    1. Load the train data:

```
f= open("train_data.txt","r")
lines = str(f.readlines())
docs1, docs2 = train_trans(lines)
f= open("train_gs.txt","r")
gs = txt_to_list(f.read())
print(gs)
```

    2. Define the train samples:

        To store our training samples we use the InputExample class. As parameters, we pass a list of sentences and a label indicating semantic similarity.

```
# Defining the train samples
train_data = list(zip(docs1, docs2))
train_samples = []
# print(train_samples)
for i in range(len(train_data)):
    train_samples.append(InputExample(texts=train_data[i], label=float(gs[i])))
```

2. **Training Parameters:**

   We then pass the training samples to the DataLoader() method which helps to shuffle (set to True) the data and produces batches of size 64.

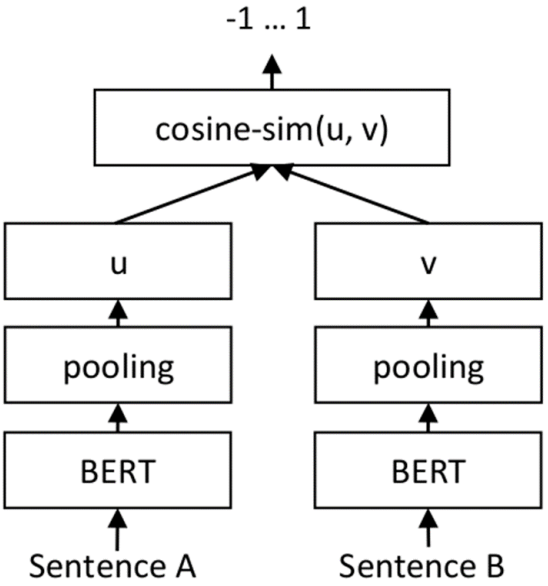
   Another parameter is the CosineSimilarityLoss() method. The loss function helps in fine-tuning the model. It determines how well our sentence embedding model would perform a task.

```
# Defining the dataloader and the train loss
train_dataloader = DataLoader(train_samples, shuffle=True, batch_size=64)

train_loss = CosineSimilarityLoss(model)
```

3. **Tuning the model:**

   To fine-tune our model to have sentence pairs annotated with a score indicating their similarity we train with a Siamese Network Architecture. This showed the sentence pairs that are similar to be close in vector space and those sentence pairs that are dissimilar to be far away in vector space.



Siamese Network Architecture

Working:

For each pair of sentences, sentence A and sentence B are passed through the Siamese network to generate the corresponding vector embeddings u and v. We have used the Cosine Similarity to compute the similarity score between these embeddings. The resultant score thus obtained is compared to gold similarity score. This technique helps the model to fine-tune and recognize the sentences which have similar meanings.

We tune the model by calling model.fit() method. We pass a list of train_objectives, which consist of tuples (dataloader, loss_function).

The model.fit() method is passed with the following parameters:

- train_objectives: We pass a list of train_objectives which consist of tuples from train_dataloader and train_loss
- epochs: is the number of passes through training dataset the model takes for training, we have set the epoch value to 5
- warmup_steps: defines the behavior scheduler. We have set the number of training steps to 100
- output_path: defines the path where the model and the test files should be saved

```
# Tuning the model
model.fit(train_objectives=[(train_dataloader, train_loss)],
          epochs=5,
          warmup_steps=100,
          output_path=model_save_path)
```

Testing:

We used 2016 STS data for testing.

```
with open('STS2016.input.headlines.txt') as f:
    lines = str(f.readlines())
s1=str(lines).split('\\n')
line1=[]
line2=[]
for i in range(0,len(s1)-1):
  line1.append(s1[i].split('\\t')[0])
  line2.append(s1[i].split('\\t')[1])
import re
string1=[]
for i in line1:
  string1.append(re.sub(r"[^a-zA-Z0-9]+",' ',i))
string2=[]
for i in line2:
  string2.append(re.sub(r"[^a-zA-Z0-9]+",' ',i))
strin=string1 +string2
```

```
import numpy as np
def cosine(u, v):
    return np.dot(u, v) / (np.linalg.norm(u) * np.linalg.norm(v))

model1 = SentenceTransformer('model_all_mp_base_v2_e10_w1000_bs64')
```

ANALYSIS OF RESULTS:

Sbert-Mpnet had the best results in Part 1 of our assignment . Below is the table comparing the results before and after fine tuning

SBERT Mpnet	**Pearson correlation before fine-tuning**	**Pearson correlation after fine-tuning**
STS2016.input.answer-answer.txt	0.74429	0.75592
STS2016.input.headlines.txt	0.84362	0.86356

STS2016.input.plagiarism.txt	0.83397	0.83984
STS2016.input.postediting.txt	0.85523	0.85851
STS2016.input.question-question.txt	0.82987	0.83805

pearson correlation scores before and after fine-tuning of SBERT

RESULT:

The Pearson Correlation scores of the five different datasets used for the SEMEVAL 2016 task are shown . The table clearly shows that after the model was adjusted, the scores increased. There is evidence of a higher pearson correlation score across all datasets.

An increase of 0.00978 in the pearson correlation was observed after fine tuning. We were able to improve the results from 0.82139 to 0.83117.

REFERENCES:

1. https://huggingface.co/
2. https://github.com/TheAtticusProject/cuad
3. https://drive.google.com/drive/u/1/folders/1Yu-JnZj1LbVBfTdPiHfMDnaKZj4eqks8