

CAUSAL EVENT EXTRACTION

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

AKULA DURGA SIREESHA (Reg. No - 40110051)

ADDANKI TEJA KUMAR (Reg. No - 40110028)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

CATEGORY – 1 UNIVERSITY BY UGC

Accredited with Grade “A++” by NAAC | 12B Status by UGC | Approved by AICTE

JEPPIAAR NAGAR, RAJIV

GANDHISALAI, CHENNAI – 600119

APRIL - 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

CATEGORY-I UNIVERSITY BY UGC

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the onafide work of **Akula Durga Sireesha** (Reg.No - 40110051) and **Addanki Teja Kumar** (Reg. No - 40110028) who carried out the Project entitled "**Causal Event Extraction**" under my supervision from November 2023 to April 2024.

Internal Guide

Dr.M.SELVI.,B.E., M.Tech., Ph.D.

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Plot No 13, Madhav Nagar, Rajiv Gandhi Salai,

Submitted for Viva voice Examination held on

Internal Examiner

External Examiner

DECLARATION

I, **AKULA DURGA SIREESHA** (Reg.No-40110051) hereby declare that the Project Report entitled "**Causal Event Extraction**" done by me under the guidance of **Dr. M. Selvi.,B.E.,M.Tech.,Ph.D.** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 13/04/2024

A.D. Sireesha

PLACE: Chennai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of Sathyabama Institute of Science and Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. M. Selvi.,B.E.,m.Tech.,Ph.D.**, for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

The goal of causal event extraction in natural language processing is finding and extracting causal links between the events or activities stated in text. The goal of this challenge is to automatically recognize cause-and-effect links, which can be helpful in a variety of applications, including knowledge graph creation and information retrieval. Advanced deep learning algorithms like LSTM and Causal BERT are needed, as well as a number of techniques including semantic role labelling. A structured representation of the causal linkages between events is the result of the causal event extraction method. Due to the richness of natural language, causal event extraction is a difficult operation, but it has the potential to offer important insights into the connections between events and actions. For causal event extraction, unsupervised algorithms can be employed since they don't need labelled training data. They might not, however, perform as well as algorithms that are trained on labelled data and are supervised. We analyze a corpus that was created by expanding the annotations of the SemEval 2010 task 8 dataset in the experiment. We concentrate on extracting all causal triplets from natural language texts, and a key component of our task is correctly identifying the tags "C" (cause), "E" (effect), and "Emb" (embedded causality), which denote the semantic roles of causal events.

TABLE OF CONTENT

CHAPTER NO	TITLE	Page No
	ABSTRACT	v
	LIST OF FIGURES	viii
1	INTRODUCTION	1
2	LITERATURE SURVEY	4
	2.1 Inferences from Literature survey	
	2.2 Open problem in Existing System	
3	REQUIREMENT ANALYSIS	8
	3.1 Risk Analysis of the Project	8
	3.2 Software Requirements Specification Document	9
4	DESCRIPTION OF PROPOSED SYSTEM	
	4.1 Proposed System	10
	4.2 Module Discription	10
	4.3 Architecture design of Proposed System	11
	4.4 Software Implementation And Testing	21
	4.5 Project Management Plan	22
5	IMPLEMENTATION DETAILS	
	5.1 Modules Used With Discription	27
	5.2 RNN Model Implementation	32
	5.3 Causal BERT Model Implementation	34
	5.4 CNN Model Implementation	37
	5.5 BERT Un-Supervised Model Implementation	39
	5.6 RNN Un-Supervised Model Implementation	41
	5.7 System Testing	43

6	RESULT AND DISCUSSION	45
	6.1 Supervised Models	46
	6.2 Un-Supervised Models	58
7	CONCLUSION	
	7.1 Future work	65
	REFERENCES	66
	APPENDIX	
	A.SOURCE CODE	68
	B.SCREENSHOT	95
	C RESEARCH PAPER	100

LIST OF FIGURES

FIGURE No.	FIGURE NAME	Page No.
1.1	DEEP LEARNING	2
4.3.1	LSTM Model	14
4.3.2	Causal BERT	15
4.3.3	RNN Model	17
4.3.4	CNN Model	18
4.3.5	BERT Model Un-Supervised	19
4.3.6	RNN Model Un-Supervised	20
4.4	System architecture diagram	22
4.5.1	Google colab	23
4.5.2	Notebook	24
4.5.3	Demo	25
5.1.1	Training Data	27
5.1.2	Testing Data	28
5.1.3	Implementation of LSTM	30
5.1.4	Implementation of LSTM	31
5.2.1	Implementation of RNN	33
5.2.2	Implementation of RNN	34
5.3.1	Implementation of Causal BERT	35
5.3.2	Implementation of Causal BERT	36
5.4.1	Implementation of CNN	38
5.4.2	Implementation of CNN	39
5.5.1	Implementation of BERT Un-Supervised	40
5.5.2	Implementation of BERT Un-Supervised	41
5.6.1	Implementation of RNN Un-Supervised	42
5.6.2	Implementation of RNN Un-Supervised	43

5.7.1	Prediction Using CNN	44
5.7.2	Prediction Using CNN	44

CHAPTER 1

INTRODUCTION

The task of determining causal linkages between events from text using natural language processing is known as causal event extraction. As it assists in identifying significant cause-and-effect relationships between events that can be used for decision-making or additional analysis, it is a crucial task in information extraction and text mining. Natural language processing methods including part-of-speech tagging, dependency parsing, and semantic role labelling are used to find the pertinent entities and connections in the text in order to perform causal event extraction. These methods aid with locating the events in the text as well as the connections that cause them to occur. We have different approaches to identify and extract causal relationships between events from text. Utilizing machine learning models, which are trained on datasets with annotations, is one method for identifying the causal connections between events. Another strategy is to employ rule-based approaches, which look for causal connections between occurrences by using established rules and patterns. To find and extract causal linkages between events in both scenarios, the models make use of methods including pattern matching, entity recognition, and relationship extraction.

Information extraction, problem solving, and decision-making are some examples of causal event extraction applications. For instance, it can be used to determine the causes of medical occurrences in order to diagnose and cure illnesses or to determine the causes of market events in order to forecast financial trends.

Understanding complex relationships, making informed judgments, condensing information, enhancing information retrieval, and enabling robots to comprehend natural language and reason about events and their causes all require causal event extraction. In the fields of medicine, finance, and social science research, it is useful.

To identify causal relationships between events and extract the relevant event pairs along with the type of causal relationship that exists between them. Causal event extraction is to develop automated methods that can accurately identify and extract causal events from large volumes of text.

DEEP LEARNING:

Deep learning is a branch of machine learning which is based on artificial neural networks. It is capable of learning complex patterns and relationships within data. In deep learning, we don't need to explicitly program everything. It has become increasingly popular in recent years due to the advances in processing power and the availability of large datasets. Because it is based on artificial neural networks (ANNs) also known as deep neural networks (DNNs). These neural networks are inspired by the structure and function of the human brain's biological neurons, and they are designed to learn from large amounts of data.

STEPS IN DEEP LEARNING:

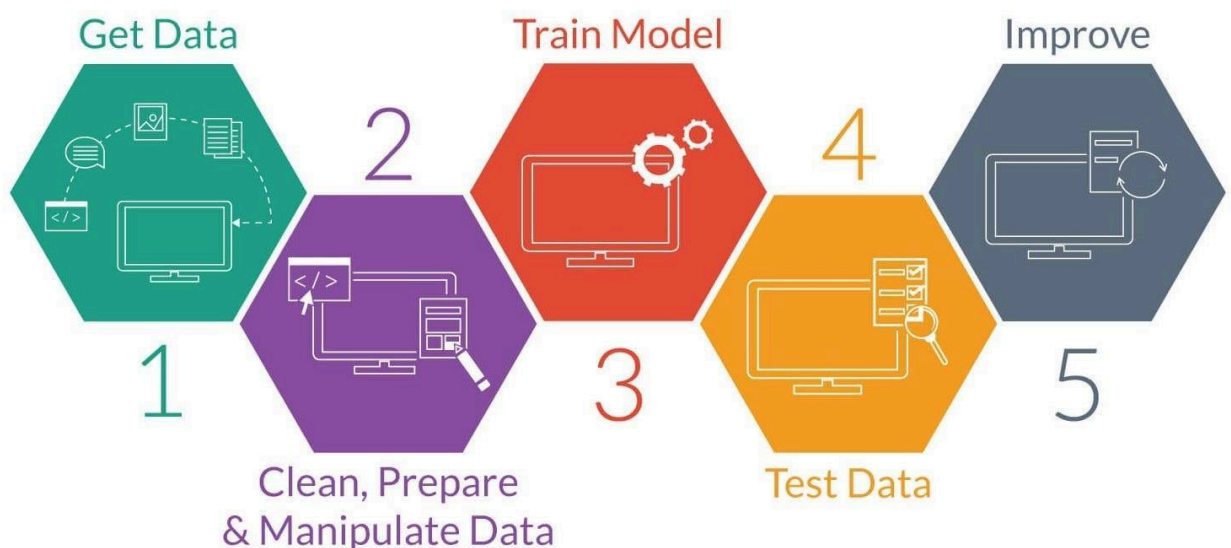


Fig. 1.1: DEEP LEARNING

TWO TYPES OF DEEP LEARNING ALGORITHMS:

1.SUPERVISED LEARNING:

Supervised learning is an approach to creating artificial intelligence (AI) where a computer algorithm is trained on input data that has been labeled for a particular output. Some of the Supervised Learning Algorithms are:

- RNN
- LSTM
- Cusal BART
- CNN

2.UNSUPERVISED LEARNING:

Unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. Some of the Unsupervised Algorithms are:

- BERT
- RNN

CHAPTER 2

LITERATURE SURVEY

Literature survey is the most important step in any kind of research. Before we start developing, we need to study the previous papers of our domain which we are working and on the basis of study we can predict or generate the drawback and start working with the reference of previous papers.

2.1 INFERENCES FROM LITRERATURE SURVEY:

Title: Causality Extraction Based on Self-Attentive BiLSTM-CRF with Transferred Embeddings Author / Year: Zhaoning Li, Qi Li, Xiaotian Zou, Jiangtao Ren /2020

Description:

In this paper, The model suggested in the paper uses a self-attentive bidirectional LSTM-CRF (Conditional Random Field) with transferred embeddings to identify causality links from text. The model uses bidirectional LSTM to model the context and self attention to collect key words and phrases. To guarantee the coherence of extracted causal links, the CRF layer is employed. To improve the quality of word embeddings, the model also uses transfer learning. The suggested model performs at the cutting edge for causality extraction on two benchmark datasets.

Title: MIMICause: Representation and automatic extraction of causal relation types from clinical notes Author / Year: Vivek Khetan , Md Imbesat Hassan Rizvi,Jessica Huber,Paige Bartusiak ,Bogdan Sacaleanu,Andrew Fano

Description:

In this paper,The dataset and model for automatically extracting causal linkages from clinical notes are presented in the study as MIMICause. The MIMIC-III database's 7,794 annotated notes make up the dataset. Convolutional neural networks (CNNs) and long short-term memory (LSTM) networks are combined in the model to uncover causal relationships between medical concepts. In order to

jointly forecast the causal relation type and direction, the model also uses a multi-task learning strategy. The evaluation findings demonstrate that the suggested model outperforms current approaches for extracting causal relations from clinical text. The suggested methodology and the MIMICause dataset have the potential to simplify clinician judgement and enhance patient outcomes

Title: Automatic Extraction of Causal Relations from Text using Linguistically Informed Deep Neural Networks. Author / Year: Tirthankar Dasgupta, Rupsa Saha, Lipika Dey and Abir Naskar / 2018

Description:

In this Paper, The research suggests a deep neural network model for automatically extracting causal links from text that is linguistically informed. To increase the precision of causal relation extraction, the model integrates syntactic and semantic data in the form of dependency trees and WordNet hypernyms. A bidirectional LSTM network is used in the model to record contextual data, and a softmax classifier is used to forecast the existence and nature of causal relationships. The suggested model outperforms current state-of-the-art methods for causal relation extraction, according to evaluation results on benchmark datasets. The concept could make a variety of applications, including text mining, information extraction, and knowledge acquisition, easier

Title: Knowledge-based extraction of adverse drug events from biomedical text Author / Year: Ning Kang, Bharat Singh, Chinh Bui, Zubair Afzal, Erik M van Mulligen & Jan A Kors / 2014

Description:

In this paper, The research provides an automatic extraction method for adverse drug events (ADEs) from biological text that is knowledge-based. To increase the accuracy of ADE extraction, the method makes use of external information sources such medical ontologies, dictionaries, and semantic networks. A named entity identification and relation extraction module is used by the system to initially identify pertinent medication and illness entities as well as their relationships. The system then uses a rule-based methodology to identify probable ADEs based on the knowledge sources and the discovered drugdisease connections. The

algorithm then uses a machine learning-based approach to categorise the discovered ADEs into several groups like severity and likelihood. The evaluation findings on benchmark datasets demonstrate that the suggested knowledge-based methodology for ADE extraction performs better than the state-of-the-art techniques. The technology may help with pharmacovigilance efforts and enhance patient safety

Title: Causality Extraction Model Based on Two-stage GCN Author / Year: Guangli Zhu , Zhengyan Sun , Shunxiang Zhang , Subo Wei , KuanChing Li / 2022

Description:

In this paper, a two-stage graph convolutional network (GCN)-based causality extraction model is proposed. The model uses GCN to capture the syntactic and semantic relationships between words in a phrase and then uses these representations to derive potential causal relationships. The putative causal links are subsequently refined by the model using a second step of GCN to weed out false positives. To increase the precision of causal relation extraction, the proposed approach also incorporates external knowledge sources in the form of medical ontologies. The proposed model outperforms current methods for causal relation extraction, as shown by evaluation results on benchmark datasets. Several applications, including pharmacovigilance, clinical decision-making, and biomedical research, may benefit from the paradigm.

2.2 OPEN PROBLEMS IN EXISTING SYSTEM:

Exploring causal relationships through causal event extraction using deep learning is an active area of research with some existing problems:

Data Annotation: Obtaining high-quality labeled data for causal event extraction is challenging. Manual annotation is time-consuming and expensive, and existing datasets might be limited in size and scope

.

Ambiguity in Causality: Causal relationships can be nuanced and context-dependent, leading to challenges in defining ground truth for training models. Ambiguity in language further complicates this issue.

Model Generalization: Deep learning models may struggle to generalize to diverse domains and languages. They often require substantial amounts of labeled data for effective generalization.

Identifying Multi-Causal Events: Some causal events involve multiple causes and effects, making it more complex to extract and represent these relationships.

Temporal Causality: Understanding causality within a temporal context, such as identifying delayed or long-term effects, remains a challenge.

Evaluating Model Trustworthiness: Ensuring that deep learning models provide reliable and interpretable causal event extraction is essential. Model trustworthiness and interpretability are ongoing concerns.

Real-time Causal Event Detection: Developing models that can extract causal events in real-time from streaming data sources is a challenging problem.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 RISK ANALYSIS OF THE PROJECT

Data Quality:

Risk: Poor data quality can lead to inaccurate models and unreliable results.

Mitigation: Thoroughly assess the quality of the dataset, including data labeling and preprocessing. Implement data cleaning and validation processes.

Model Complexity:

Risk: Deep learning models can be complex and challenging to train and optimize.

Mitigation: Plan for a step-by-step model development process, starting with simpler models and gradually increasing complexity. Use appropriate architectures and hyperparameter tuning.

Overfitting:

Risk: Deep learning models may overfit the training data, leading to poor generalization.

Mitigation: Implement techniques like dropout, early stopping, and regularization to prevent overfitting. Monitor model performance on validation data.

Computational Resources:

Risk: Deep learning often requires significant computational resources, which can be expensive.

Mitigation: Assess and plan for the necessary hardware and cloud resources. Consider cost-effective alternatives, such as using cloud-based GPU instances.

Data Privacy and Security:

Risk: Handling sensitive data may pose security and privacy risks.

Mitigation: Implement strong data security measures, including encryption and access controls. Ensure compliance with relevant data privacy regulations.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

HARDWARE REQUIREMENTS

System : i5 Processor
Hard Disk : 512 GB.
Input Devices : Keyboard, Mouse
Ram : 8 GB.

SOFTWARE REQUIREMENTS

Operating system : Windows
Coding Language : Python, VS Code, Goggle colab.

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

4.1 PROPOSED SYSTEM

- Information, enhancing information retrieval, and enabling robots to comprehend natural language and reason about events and their causes all require causal event extraction. In the fields of medicine, finance, and social science research, it is useful.
- To identify causal relationships between events and extract the relevant event pairs along with the type of causal relationship that exists between them. Causal event extraction is to develop automated methods that can accurately identify and extract causal events from large volumes of text.

4.2 ADVANTAGES OF PROPOED SYSTEM

Improved Accuracy: Deep learning models have the potential to significantly improve the accuracy of causal event extraction by learning complex patterns and relationships in textual data.

Scalability: Deep learning models can handle large datasets efficiently, making it possible to scale up causal event extraction to cover a wide range of sources.

Adaptability: These models can be fine-tuned for specific domains or use cases, increasing their adaptability to different applications.

Reduced Human Bias: By relying on automated deep learning algorithms, potential human biases in causal event extraction can be minimized.

Real-time Analysis: Deep learning models can be implemented for real-time or near-real-time causal relationship extraction, allowing for timely insights from streaming data.

4.3 MODULE DESCRIPTION:

LSTM Model Implementation:

The implementation of the LSTM model first it takes the input and does the tokenization for the extracted words and then does the Glove embedding means basically pre defined weights of the words these words have been trained by the wikipedia data or many other data sets these glove embedding has been defined. Then the data is classified and predicted. LSTM is the same as the RNN but in the LSTM it will remember the weights but in the RNN it has the forget gate so it does not remember the before weights of the previous words.

RNN Model Implementation:

Talks about the implementation of the RNN model using methods like glove embedding, we first preprocess the text input by turning the words into numerical vectors. The (RNN) layer, which is made to handle sequential data, is then applied to the list of word vectors. At each time step, the RNN layer updates its hidden state by taking into account both the current input and the prior hidden state. As a result, the context and relationships between words in a phrase can be captured by the model.

To extract more intricate features from the hidden state after the RNN layer, we can add additional layers such as dense or dropout layers. Finally, we may output the predicted class probabilities for the text using a dense layer with a softmax activation function.

Causal BERT Model Implementation:

Talks about the implementation of the Causal BERT model first uses the BERT tokenizer to sub word-tokenize the text data as part of preprocessing. After that, we run a Causal BERT layer across the tokenized sequence of subword units. Each token solely attends to its left context in the left-to-right input sequence of the Causal BERT layer. As a result, the temporal connections between words in a sentence can be captured by the model. After adding more layers, such as dense, to extract more complex properties from the hidden state, we utilise a final dense layer with a softmax activation function to output the projected class probabilities for the text.

CNN Model Implementation:

Talks about the Implementation of the CNN model. The word vector sequence is first preprocessed, and then the (CNN) layer is used in order to extract local characteristics from the input sequence. By sliding filters of a predetermined size over the list of word vectors, the CNN layer computes a dot product between the filter weights and the local window of words at each location. In order to extract a variety of local features from the input sequence, this step is repeated with various filters.

Then, using a final dense layer with a softmax activation function, we can output the predicted class probabilities for the text. Additional layers, such as pooling and dense layers, can be added to extract more complicated characteristics from the extracted local features.

BERT Unsupervised Model Implementation:

Talks about the implementation of BERT unsupervised. We begin by preparing the text data by tokenizing it into subword units using the BERT tokenizer in order to do unsupervised text classification. The pre-trained BERT model is then applied to the tokenized subword unit sequences, producing a sequence of contextualized embedding's for each token.

Then, using the contextualized embedding's of the tokens, we employ clustering techniques like K-means or hierarchical clustering to divide the tokens into a predetermined number of clusters. This enables us to find word groups in the input text that are semantically related. Finally, we train and forecast the output using the cluster assignments as features.

RNN Unsupervised Model Implementation:

Talks about the implementation of RNN unsupervised. Sequential relationships between words in a sentence are captured by the RNN layer. To encode the input text and decode it to come back to the original text, we employed an auto encoder. Reconstruction loss allows for the unsupervised training of this. The encoded vectors can be organized into a predetermined number of clusters using clustering techniques after the RNN-based auto encoder has learned a valid encoding of the input text. This enables us to find groupings of texts that share semantic similarities.

Finally, depending on the most prevalent cluster labels, we can give cluster labels to each document. Where in papers with comparable word distributions are clustered. And the output is predicted.

RNN Unsupervised Model Implementation:

Talks about the implementation of RNN unsupervised. Sequential relationships between words in a sentence are captured by the RNN layer. To encode the input text and decode it to come back to the original text, we employed an auto encoder. Reconstruction loss allows for the unsupervised training of this. The encoded vectors can be organized into a predetermined number of clusters using clustering techniques after the RNN-based auto encoder has learned a valid encoding of the input text. This enables us to find groupings of texts that share semantic similarities.

Finally, depending on the most prevalent cluster labels, we can give cluster labels to each document. Where in papers with comparable word distributions are clustered. And the output is predicted.

LSTM Model:

Input layer: The preprocessed text data, which is commonly represented as a series of words or tokens, is ingested by the LSTM architecture's first layer. The layer in the design is an embedding layer, which transforms the text data input into numerical representations called embedding's. The semantic meaning of the words and their connections to other words in the input sequence are captured by

embedding's. The LSTM layer analyses the input sequence word-by-word and keeps an internal state that compiles the data seen up to this point. Each memory cell in the LSTM layer has the capacity to retain knowledge for a protracted period of time.

LSTM architecture's output layer can take on a variety of shapes depending on the precise task of causal event extraction. For instance, the output layer can forecast whether a specific event will be causative or not, or it can forecast the causal chain connecting two occurrences. The LSTM architecture is often trained using supervised learning methods, in which labelled data is utilised to optimise the model's parameters in order to minimise a certain loss function.

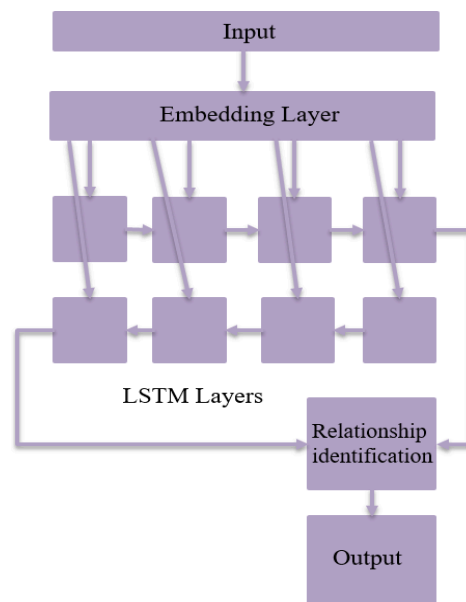


Fig. 4.3.1: LSTM Model

Causal BERT Model:

Text data is often preprocessed and represented as a series of tokens, such as words or sub words, at the input layer. The embedding layer converts each token into a high- dimensional vector representation that includes both its semantic and contextual information. The transformer architecture, which consists of numerous layers of feedforward and self-attention neural networks, is used to do this.

In the Causal BERT model, the input sequence is causally masked to make sure that the model only uses data from the past to predict the current token. This makes it easier for the model to depict the causes of various events. A fixed-length representation of the input text data is created by the pooling layer, which condenses the token-level embedding's. Different pooling techniques, such as mean pooling and maximum pooling, can be used to accomplish this.

Using the information gleaned from the pooling layer, the fully connected layers combine the input text data to develop high-level representations of it. For the purpose of preventing overfitting, these layers may additionally include dropout or regularisation. The Causal BERT model uses supervised learning to fine-tune the pre- trained weights of the embedding layer and the fully linked layers on a downstream task of causal event extraction.

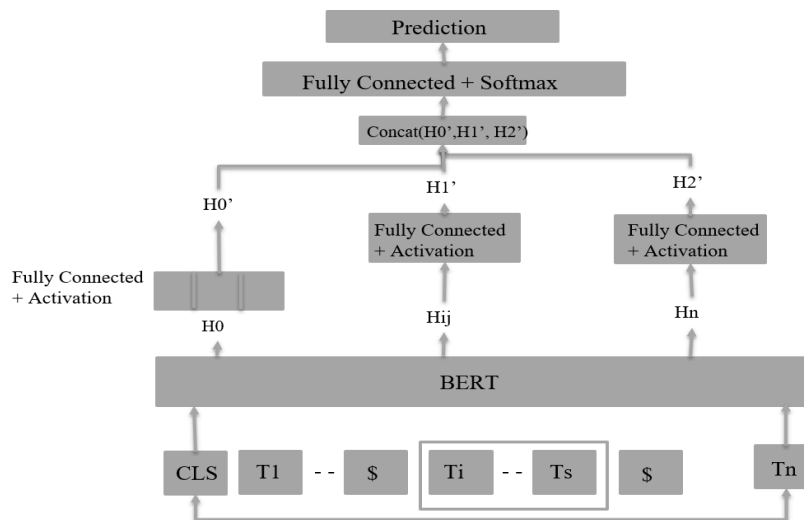


Fig. 4.3.2: Causal BERT

RNN Model:

The first layer of the RNN architecture accepts preprocessed text data, which is commonly represented as a sequence of words or tokens. The next layer of the design is an embedding layer, which turns the input text data into numerical representations known as embeddings. In the input sequence, the words' relationships to one another and their semantic meaning are captured via embeddings.

The RNN layer analyses the input sequence one word at a time and keeps an internal state that compiles the knowledge gained thus far. Recurrent connections that pass information from one time step to the next make up the RNN layer. Depending on the exact purpose of causal event extraction, the RNN architecture's output layer can take on a variety of shapes. For instance, the output layer can forecast whether a specific event will be causative or not, or it can forecast the causal chain connecting two occurrences.

The RNN architecture is often trained using supervised learning approaches, where labelled data is utilised to optimise the model's parameters in order to minimise a certain loss function. RNNs are more prone to the vanishing gradient problem than LSTM-based systems, which may limit their ability to detect long-term dependencies.

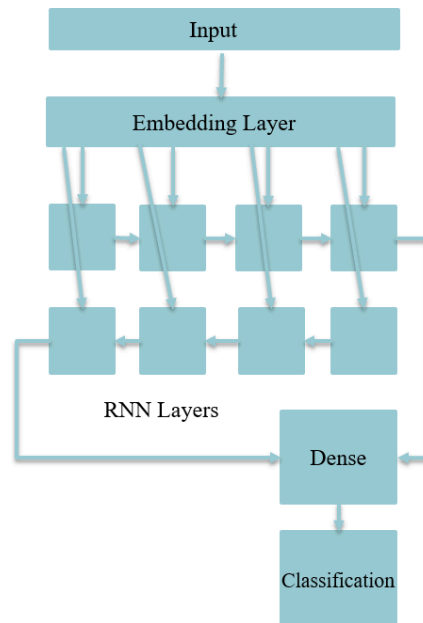


Fig. 4.3.3: RNN MODEL

CNN Model:

Preprocessing is required in order to turn the text data into a numerical representation that the CNN can use. Tokenization, stemming, and building a vocabulary of every unique term in the dataset are frequently used in this process. After the text is converted to numbers, it is frequently sent through a layer of embedding to produce a dense representation of the text that the CNN may learn from. This can lessen the input's dimensionality and aid in capturing semantic links between words.

After that, one or more 1D convolutional layers are fed the embedded text. In order to extract features from the input word sequence, each convolutional layer comprises a set of filters that convolve over the words. These filters are capable of detecting n-gram patterns within the input sequence, where n is the filter's breadth.

To transform the series of output vectors into a fixed-length vector representation of the full input text, you can omit the convolutional layer(s) and instead add pooling layers like max-pooling or average-pooling. To classify the text into one of several potential categories, the fixed-length vector representation is finally run through a dense output layer with a softmax activation function. To minimise the difference between the expected and actual output labels, the complete model is then trained using backpropagation and gradient descent.

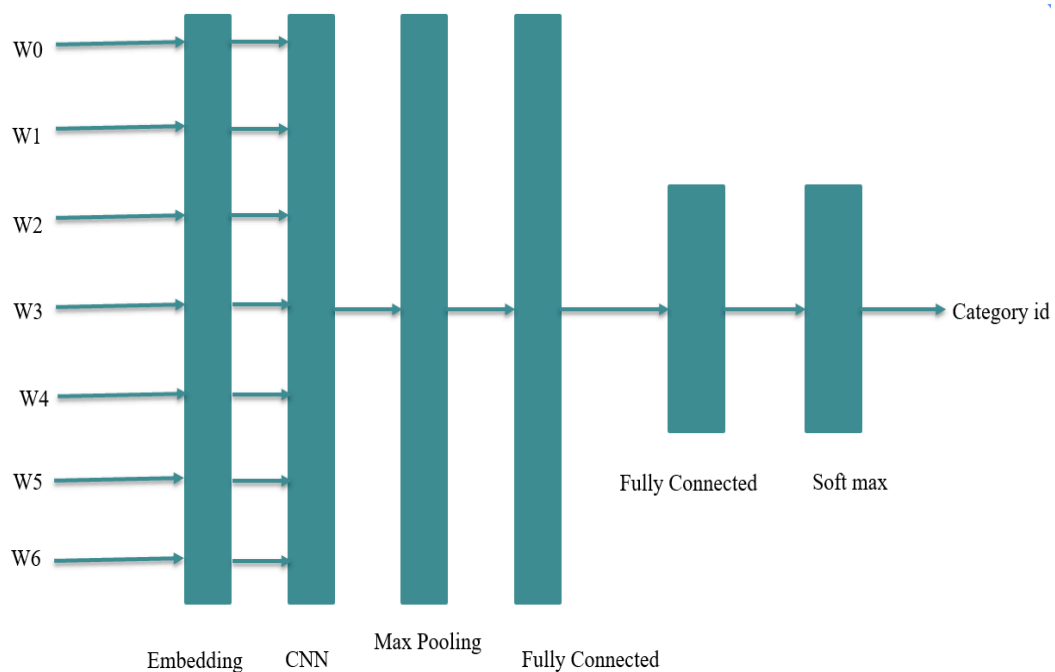


Fig. 4.3.4: CNN Model

Unsupervised BERT Model:

Masked Language Model (MLM): The initial phase entails training a BERT model on a sizable corpus of unlabeled text data using the MLM objective. By doing so, the model is able to capture the links between the words in the input sequence and learn a robust representation of the language.

The next stage is to utilize the trained model to extract features from the input text data once the BERT model has been trained on the unlabeled data using MLM. To accomplish this, the input text is run through the BERT model to extract the contextualized embedding's of the input sequence's tokens. The next step is to group the contextualized embedding's that were extracted from the BERT model according to how similar they are. This facilitates the collection of contextualized embedding's corresponding to related events.

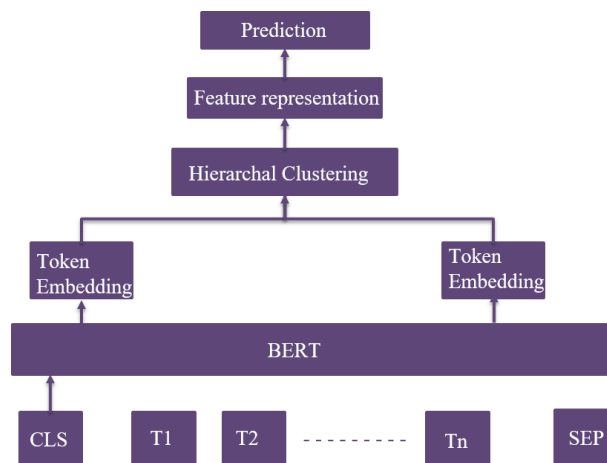


Fig. 4.3.5: BERT(Unsupervised) Model

Unsupervised RNN Model:

The incoming text data must first be transformed into a numerical representation in the first stage. Word embeddings, which represent words as high-dimensional vectors, are created to do this. Following the creation of the word embeddings, they must be run through an RNN encoder. The RNN encoder maintains an internal state that summarises the data seen thus far as it reads in the word embeddings one at a time.

After utilising the RNN encoder to create encoded representations of the input text data, the next step is to group the encoded representations according to how similar they are. This makes it easier to aggregate together encoded representations of related events. Identification of the causal linkages between the events comes next once the encoded representations have been clustered. The causal relationships between various occurrences can be ascertained by employing methods like temporal order analysis or causal inference algorithms. RNNs are especially good at capturing the temporal relationships between the words in the input sequence, which makes them suitable for applications requiring the extraction of causal events where the temporal order of events is crucial.

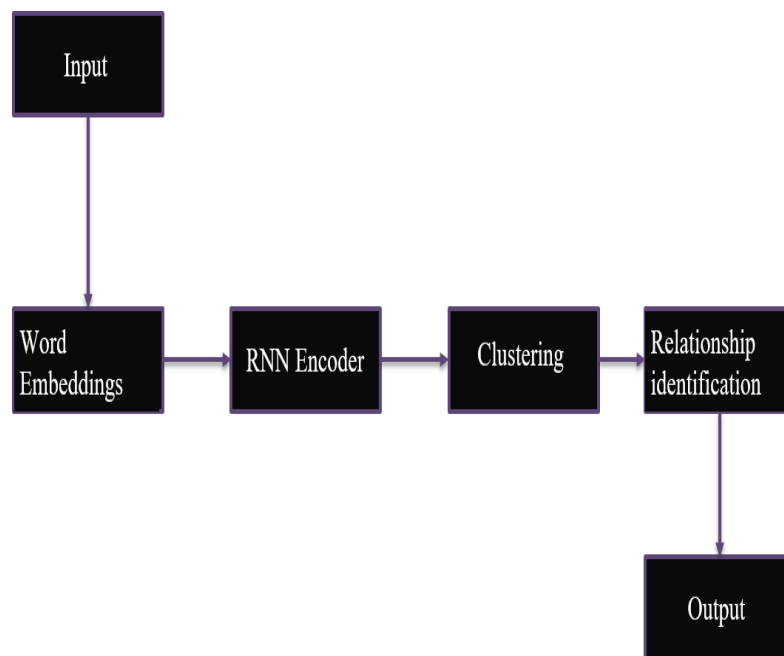


Fig. 4.3.6: Unsupervised RNN Model

4.4 ARCHITECTURE/OVERALL DESIGN OF PROPOSED SYSTEM

SYSTEM ARCHITECTURE:

The whole data set has been divided into the testing and training. The data is preprocessed from the data set like Tokenization, Removing the stop words etc. After the preprocessing we have started building the models, Here in our project we have

built 6 models. 4 models on the supervised data. And the 2 models on the unsupervised data.

In the supervised models we have built the RNN, LSTM, Causal BERT and CNN. In unsupervised models we have built the BERT and RNN. Then we have compared all models accuracy and the model which we have got more accuracy using that model we have predicted the output. In the above all models CNN has more accuracy comparing with all models.

So using the CNN model we have predicted the output. In the output if we give the sentence with the entities the model will predict the relationship between the two entities. All the models has been clearly explained below with the architecture.

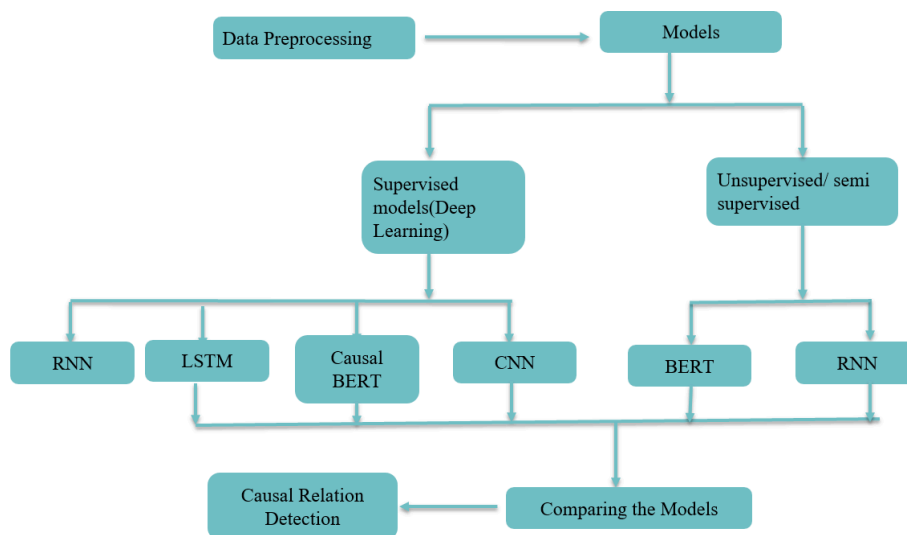


Fig. 4.4.1: SYSTEM ARCHITECTURE DIAGRAM

4.5 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL/SYSTEM

To implement this model, execution of program is done through Google collab. Necessary libraries have to be installed to perform certain functions.

GOOGLE COLAB:

linkdone. Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

Features of Google colab:

- Write and execute code in Python
- Document your code that supports mathematical equations
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Import external datasets e.g. from Kaggle
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

Download the Google colab installer for Windows

1. Open a web browser.
2. Go to the Google Colab website: <https://colab.research.google.com/>
3. Sign in with your Google account.
4. You can create a new notebook by clicking on "File" and then selecting "New Notebook." You can also open existing notebooks or import notebooks from Google Drive.
5. You can run code cells, install libraries, and work on your Python projects within the Colab environment.

Install Google colab on Windows:

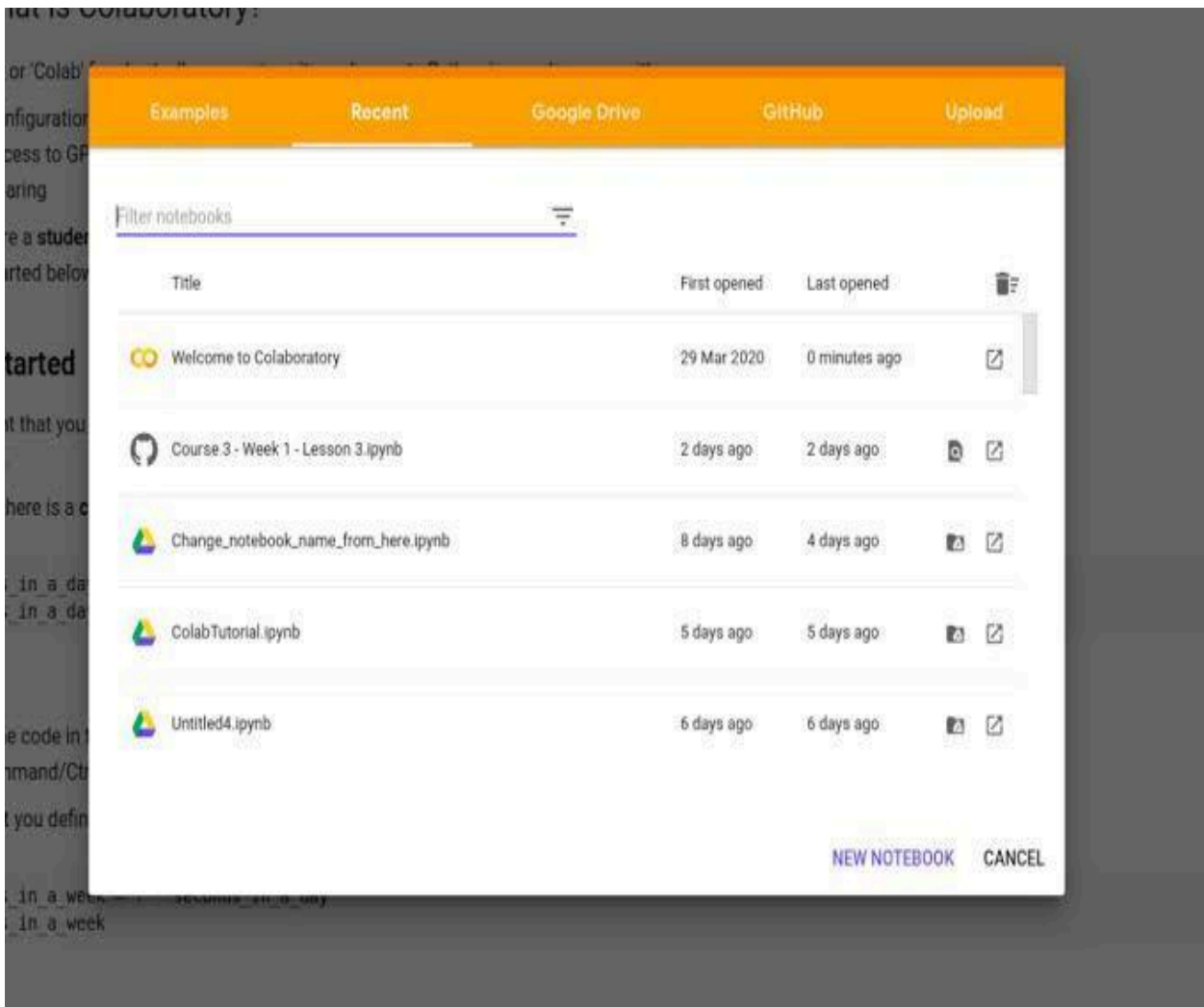


Fig. 4.5.1: GOOGLE COLAB

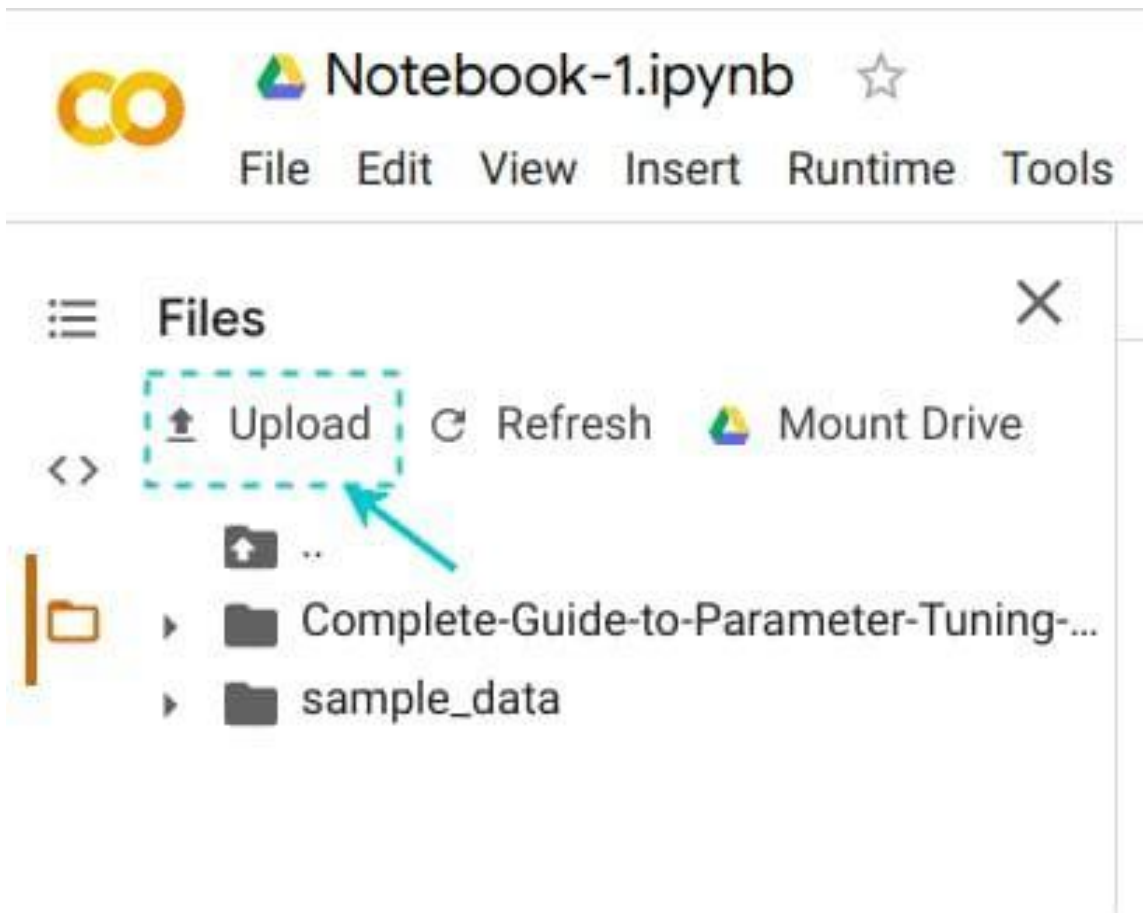


Fig. 4.5.2: NOTEBOOK

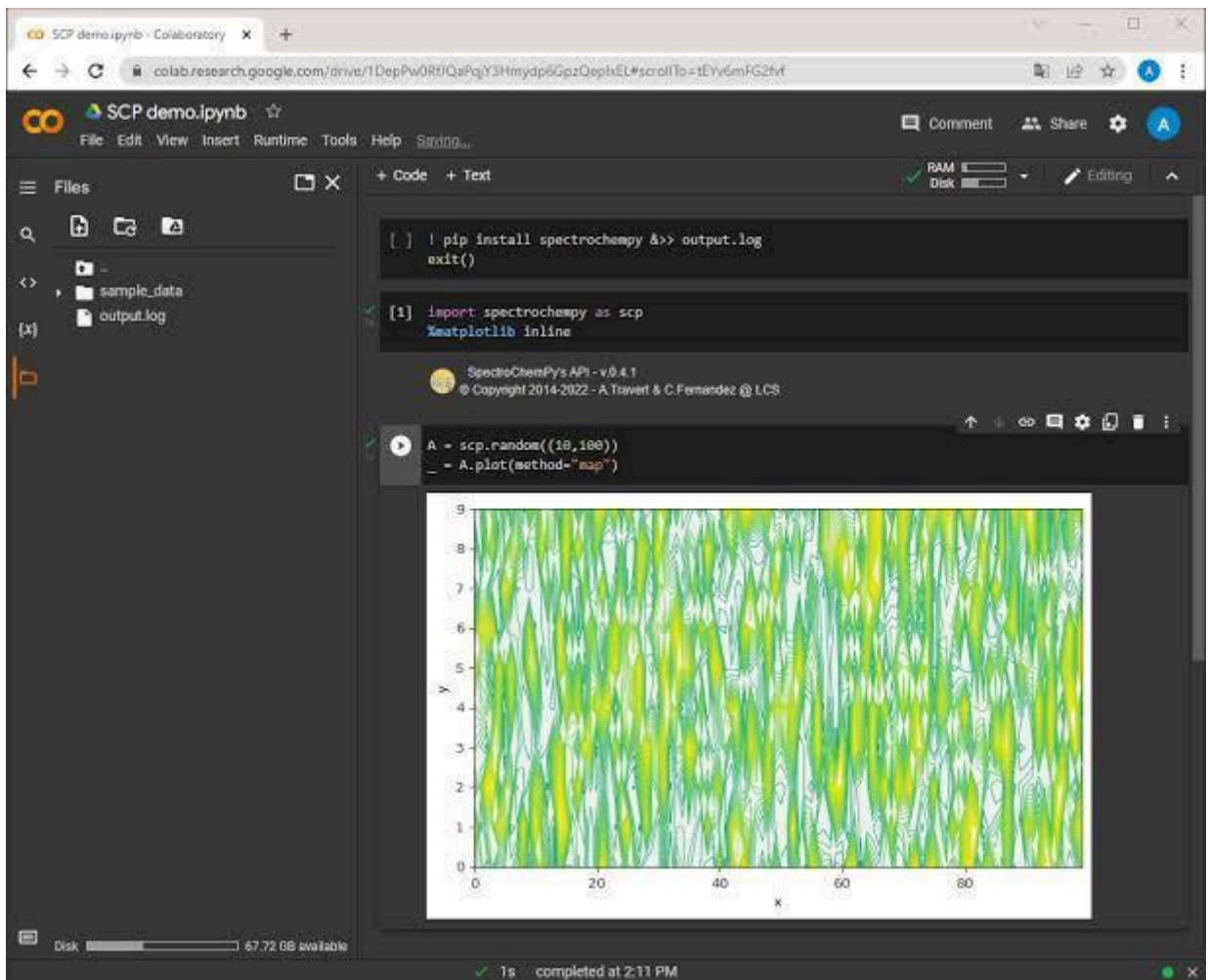


Fig: 4.5.3: DEMO

4.6 PROJECT MANAGEMENT PLAN

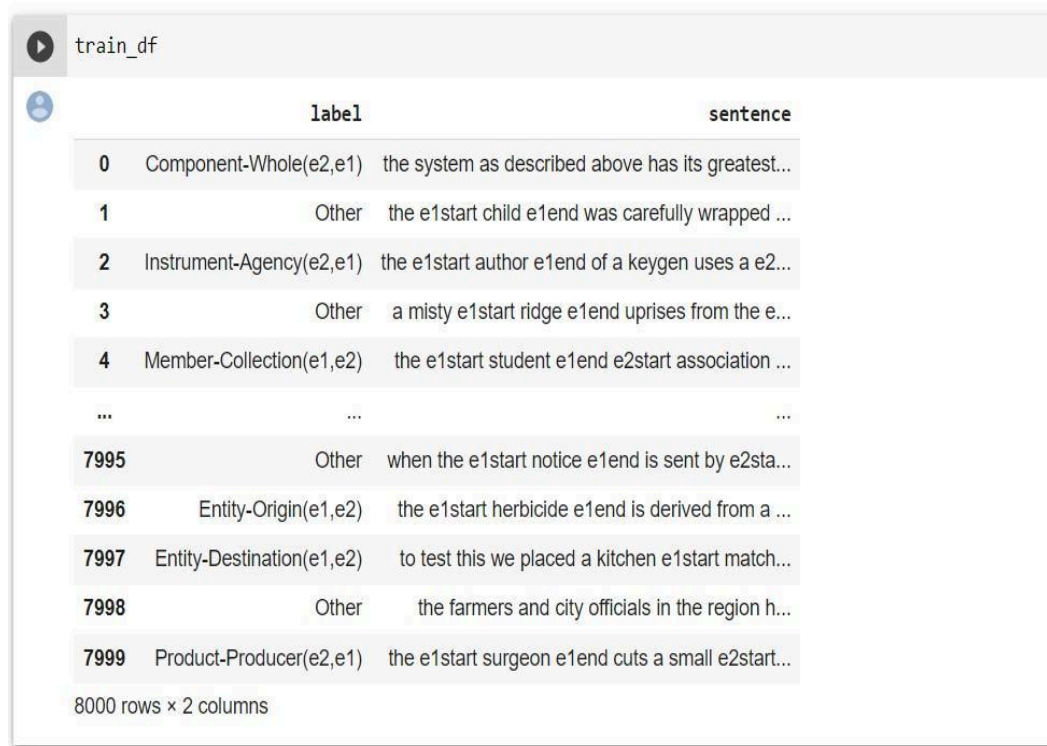
August	Literature survey
September	Data acquisition
October	Data preprocessing
Novemeber	Training and Splitting
December	Loading, training and testing the model.
January	Predicting the output and generating the final report

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 Modules used with description:

Figures 5.1 and 5.2 Talks about the test data and the train data is used to test and train the Models.



	label	sentence
0	Component-Whole(e2,e1)	the system as described above has its greatest...
1	Other	the e1start child e1end was carefully wrapped ...
2	Instrument-Agency(e2,e1)	the e1start author e1end of a keygen uses a e2...
3	Other	a misty e1start ridge e1end uprises from the e...
4	Member-Collection(e1,e2)	the e1start student e1end e2start association ...
...	...	...
7995	Other	when the e1start notice e1end is sent by e2sta...
7996	Entity-Origin(e1,e2)	the e1start herbicide e1end is derived from a ...
7997	Entity-Destination(e1,e2)	to test this we placed a kitchen e1start match...
7998	Other	the farmers and city officials in the region h...
7999	Product-Producer(e2,e1)	the e1start surgeon e1end cuts a small e2start...

8000 rows × 2 columns

Fig. 5.1.1: TRAINING DATA



	label	sentence
0	Message-Topic(e1,e2)	the most common e1start audits e1end were abou...
1	Product-Producer(e2,e1)	the e1start company e1end fabricates plastic e...
2	Instrument-Agency(e2,e1)	the school e1start master e1end teaches the le...
3	Entity-Destination(e1,e2)	the suspect dumped the dead e1start body e1end...
4	Cause-Effect(e2,e1)	avian e1start influenza e1end is an infectious...
...	...	...
2712	Instrument-Agency(e2,e1)	after seating all the idols which itself takes...
2713	Product-Producer(e1,e2)	the minister attributed the slow production of...
2714	Component-Whole(e2,e1)	the e1start umbrella e1end e2start frame e2end...
2715	Product-Producer(e1,e2)	manos the hands of fate is a lowbudget horror ...
2716	Entity-Destination(e2,e1)	a few days before the service tom burris had t...

2717 rows × 2 columns

Fig.5.1.2: TESTING DATA

LSTM Model Implementation:

From Figure 5.3 and 5.4 explains about the implementation of the LSTM model first it takes the input and does the tokenization for the extracted words and then does the Glove embedding means basically pre-defined weights of the words these words have been trained by the Wikipedia data or many other data sets these glove embedding has been defined. Then the data is classified and predicted. LSTM is the same as the RNN but in the LSTM it will remember the weights but in the RNN it has the forget gate so it does not remember the before weights of the previous words. Implementing an LSTM (Long Short-Term Memory) model for causal event extraction involves several steps. Here's a general outline of how you might approach it:

5.1.1 Data Preparation:

Obtain a dataset containing text documents annotated with causal events. Preprocess the text data by tokenizing it into words or subwords, removing stop words, punctuation, etc. Convert the text data into numerical format suitable for input into the LSTM model. This can be achieved through techniques like word embeddings (e.g., Word2Vec, GloVe) or character-level embeddings.

5.1.2 Feature Representation:

Represent your text data in a format suitable for input to the LSTM model. This could be word embeddings like Word2Vec or GloVe, or you could use character-level embeddings. Pad or truncate sequences to ensure they are of equal length.

5.1.3 Evaluation:

Calculate metrics such as accuracy, precision, recall, and F1-score to evaluate the performance of your LSTM model. Additionally, consider using techniques like cross-validation for more robust evaluation.

5.1.4 Training:

Split your dataset into training, validation, and testing sets. Train the LSTM model on the training data. During training, adjust the model's weights to minimize a suitable loss function, such as binary cross-entropy. Monitor the model's performance on the validation set to avoid overfitting. You may need to adjust hyperparameters like learning rate, dropout rate, etc., to optimize performance.

```
class RelationClassification_Model2(nn.Module):

    def __init__(self, embedding_size, hidden_size, vocab_size, num_layers, class_num, pretrained_embedding, pad_idx):
        super(RelationClassification_Model2, self).__init__()

        self.hidden_size = hidden_size
        self.embedding = nn.Embedding.from_pretrained(pretrained_embedding, freeze = True, padding_idx = pad_idx)
        self.lstm = nn.LSTM(embedding_size, hidden_size, num_layers, bidirectional = True)
        self.linear = nn.Linear(2*hidden_size, class_num)
        self.softmax = nn.LogSoftmax(dim=1)
        self.dropout = nn.Dropout(0.7)

    def forward(self, X, mask_e1, mask_e2):
        embeddings = self.dropout(self.embedding(X))
        hiddens, _ = self.lstm(embeddings)
        x = hiddens.view((hiddens.shape[0], hiddens.shape[1], 2, self.hidden_size))
        e1, e2 = max_avg_pooling(x, mask_e1, mask_e2)
        concats = torch.cat((e1, e2), dim = 1)

        outputs = self.softmax(self.linear(concats))

        return outputs

[ ] def max_avg_pooling(x, mask_e1, mask_e2):
    e1 = torch.zeros((x.shape[1], x.shape[3])).to(device)
    e2 = torch.zeros((x.shape[1], x.shape[3])).to(device)

    for i in range(x.shape[1]):
        sample = x[:, i, :, :].to(device)
        y = sample[mask_e1[:, i]].to(device)
        avg = torch.zeros((1, x.shape[3])).to(device)
        for j in range(y.shape[0]):
            avg += (torch.maximum(y[j, 0, :], y[j, 1, :]) - avg).to(device) / (j + 1)
        e1[i, :] = avg

        y = sample[mask_e2[:, i]].to(device)
        avg = torch.zeros((1, x.shape[3])).to(device)
        for j in range(y.shape[0]):
            avg += (torch.maximum(y[j, 0, :], y[j, 1, :]) - avg).to(device) / (j + 1)

        e2[i, :] = avg
```

Fig.5.1.3: IMPLEMENTATION OF LSTM

5.1.5 Model Architecture:

Design the architecture of the LSTM model. This typically involves stacking one or more LSTM layers followed by a dense layer for classification.

Decide on the input representation. This could be word embeddings, character embeddings, or a combination of both. Define the output layer. Since this is a causal event extraction task, the output layer should have binary classification (causal or non-causal) for each token in the input sequence.

This is a basic outline, and you may need to adapt and extend it based on your

specific requirements and data characteristics. Additionally, you might explore more sophisticated architectures, such as incorporating attention mechanisms or utilizing pre-trained language models like BERT for further improvement.

```

torch.manual_seed(2045)

int_2_cat_class = dict(enumerate(train_df['label'].cat.categories))

# Training and validation Data Generators
training_set = SemEval2010_task8_Dataset2(train_df,partition['train'],vocab)

pad_idx = 1917494

# Parameters
params = {'batch_size': 256,
          'shuffle': True,
          'num_workers':2,
          'collate_fn' : BatchPadCollate2(pad_idx=pad_idx)
        }

training_generator = torch.utils.data.DataLoader(training_set, **params)

validation_set = SemEval2010_task8_Dataset2(train_df,partition['validation'],vocab)
validation_generator = torch.utils.data.DataLoader(validation_set, **params)

# Hyperparameters
embedding_size = 300
hidden_size = 150
vocab_size = len(vocab_df)
num_layers = 2
class_num = len(int_2_cat_class)
learning_rate = 5e-3
num_epochs = 15

# Model
model3 = RelationClassification_Model2(embedding_size, hidden_size, vocab_size, num_layers,class_num, embedding,pad_idx).to(device)

# Define Loss function and Optimizer
criterion = nn.NLLLoss(ignore_index = pad_idx)
optimizer = optim.Adam(model3.parameters(), lr = learning_rate, weight_decay= 1e-3)
# scheduler = MultiStepLR(optimizer, milestones=[30,80], gamma=0.5)

train_losses3 = []
validation_losses3 = []

train_accs3 = []
val_accs3 = []

```

Fig.5.1.4: IMPLEMENTATION OF LSTM

5.2 RNN Model Implementation.

Figures 5.5 and 5.6 Talks about the implementation of the RNN model using methods like glove embedding, we first preprocess the text input by turning the words into numerical vectors. The (RNN) layer, which is made to handle sequential data, is then applied to the list of word vectors. At each time step, the RNN layer updates its hidden state by taking into account both the current input and the prior hidden state. As a result, the context and relationships between words in a phrase can be captured by the model.

To extract more intricate features from the hidden state after the RNN layer, we can add additional layers such as dense or dropout layers. Finally, we

may output the predicted class probabilities for the text using a dense layer with a softmax activation function.

5.2.1 Data Preparation:

Obtain a dataset containing text documents annotated with causal events. Preprocess the text data by tokenizing it into words or subwords, removing stop words, punctuation, etc. Convert the text data into numerical format suitable for input into the RNN model. This can be achieved through techniques like word embeddings (e.g., Word2Vec, GloVe) or character-level embeddings.

5.2.2 Model Architecture:

Design the architecture of the RNN model. This typically involves using a single layer of recurrent cells (such as SimpleRNN) followed by a dense layer for classification. Decide on the input representation. This could be word embeddings, character embeddings, or a combination of both. Define the output layer. Since this is a causal event extraction task, the output layer should have binary classification (causal or non-causal) for each token in the input sequence.

```

torch.manual_seed(2045)

int_2_cat_class = dict(enumerate(train_df['label'].cat.categories))

# Training and validation Data Generators
training_set = SemEval2010_task8_Dataset2(train_df,partition['train'],vocab)

pad_idx = 1917494

# Parameters
params = {'batch_size': 256,
          'shuffle': True,
          'num_workers':2,
          'collate_fn' : BatchPadCollate2(pad_idx=pad_idx)
        }

training_generator = torch.utils.data.DataLoader(training_set, **params)

validation_set = SemEval2010_task8_Dataset2(train_df,partition['validation'],vocab)
validation_generator = torch.utils.data.DataLoader(validation_set, **params)

# Hyperparameters
embedding_size = 300
hidden_size = 150
vocab_size = len(vocab_df)
num_layers = 2
class_num = len(int_2_cat_class)
learning_rate = 5e-3
num_epochs = 15

# Model
model4 = RelationClassification_Model2(embedding_size, hidden_size, vocab_size, num_layers,class_num, embedding,pad_idx).to(device)

# Define Loss function and Optimizer
criterion = nn.NLLLoss(ignore_index = pad_idx)
optimizer = optim.Adam(model4.parameters(), lr = learning_rate, weight_decay= 1e-3)
# scheduler = MultiStepLR(optimizer, milestones=[30,80], gamma=0.5)

train_losses4 = []
validation_losses4 = []

train_accs4 = []
val_accs4 = []

```

Fig. 5.2.1: IMPLEMENTATION OF RNN

5.2.3 Training:

Split your dataset into training, validation, and testing sets. Train the RNN model on the training data. During training, adjust the model's weights to minimize a suitable loss function, such as binary cross-entropy. Monitor the model's performance on the validation set to avoid overfitting. You may need to adjust hyperparameters like learning rate, dropout rate, etc., to optimize performance.

```

loss.backward()
optimizer.step()

# print statistics
running_trainloss += loss.item()
running_loss += loss.item()

train_acc += torch.sum(torch.argmax(batch_train_outputs, dim =1) == batch_train_labels.long()) / torch.numel(batch_train_labels)

if train_cnt % 50 == 49: # print every 10 mini-batches
    print("Cost of Train data after %i iterations in epoch %i : %f" %(train_cnt + 1,epoch + 1, running_loss / 50))
    running_loss = 0.0

train_cnt += 1

# Validation
running_loss, val_acc = 0.0, 0.0

with torch.set_grad_enabled(False):
    for batch_data, batch_labels,mask_e1,mask_e2 in tqdm(validation_generator):
        # Transfer to GPU
        batch_data, batch_labels,mask_e1,mask_e2 = batch_data.to(device), batch_labels.to(device),mask_e1.to(device),mask_e2.to(device)

        # Model computations
        # forward + backward + optimize
        batch_outputs = model3(batch_data,mask_e1,mask_e2 )

        val_loss = criterion(batch_outputs, batch_labels)

        running_valloss += val_loss.item()
        running_loss += val_loss.item()

        val_acc += torch.sum(torch.argmax(batch_outputs, dim =1) == batch_labels.long()) / torch.numel(batch_labels)

    if val_cnt % 10 == 9: # print every 10 mini-batches
        print("Cost of Validation data after %i iterations in epoch %i : %f" %(val_cnt + 1,epoch + 1, running_loss / 10))
        running_loss = 0.0

    val_cnt += 1

train_losses4.append(running_trainloss / train_cnt)
train_accs4.append(train_acc / train_cnt * 100)
validation_losses4.append(running_valloss / val_cnt)
val_accs4.append(val_acc / val_cnt * 100)

```

Fig. 5.2.2: IMPLEMENTATION OF RNN

5.3 Causal BERT Model Implementation

Figures 5.7 and 5.8 talks about the implementation of the Causal BERT model first uses the BERT tokenizer to sub word-tokenize the text data as part of preprocessing. After that, we run a Causal BERT layer across the tokenized sequence of subword units. Each token solely attends to its left context in the left-to-right input sequence of the Causal BERT layer. As a result, the temporal connections between words in a sentence can be captured by the model. After adding more layers, such as dense, to extract more complex properties from the hidden state, we utilise a final dense layer with a softmax activation function to output the projected class probabilities for the text.

5.3.1 Data Preparation:

Obtain a dataset containing text documents annotated with causal events. Preprocess the text data by tokenizing it into tokens or subwords compatible with BERT. Convert the text data into the required format for fine-tuning BERT, typically token IDs, segment IDs, and attention masks.

```
)

eval_loss = eval_loss / nb_eval_steps
results = {"loss": eval_loss}
preds = np.argmax(preds, axis=1)
write_prediction(
    self.args, os.path.join(self.args.eval_dir, "proposed_answers.txt"), preds
)

result = compute_metrics(preds, out_label_ids)
results.update(result)

print("Eval results")
for key in sorted(results.keys()):
    print(" {} = {:.4f}".format(key, results[key]))

return results, loss_eval, preds, out_label_ids

def save_model(self):
    if not os.path.exists(self.args.model_dir):
        os.makedirs(self.args.model_dir)
    model_to_save = (
        self.model.module if hasattr(self.model, "module") else self.model
    )
    model_to_save.save_pretrained(self.args.model_dir)
    torch.save(self.args, os.path.join(self.args.model_dir, "training_args.bin"))
    logger.info("Saving model checkpoint to %s", self.args.model_dir)
def load_model(self):
    if not os.path.exists(self.args.model_dir):
        raise Exception("Model doesn't exists! Train first!")

    try:
        self.args = torch.load(
            os.path.join(self.args.model_dir, "training_args.bin")
        )
        self.config = self.config_class.from_pretrained(self.args.model_dir)
        self.model = self.model_class.from_pretrained(
            self.args.model_dir, config=self.config, args=self.args
        )
        self.model.to(self.device)
        logger.info("Model Loaded")
    except:
        raise Exception("Some files might be missing...")
```

Fig. 5.3.1: IMPEMNTATION OF CAUSAL BERT

5.3.2 Model Fine-tuning:

Load a pre-trained BERT model suitable for your task (e.g., BERT-base, BERT-large). Add task-specific layers on top of the pre-trained BERT model. For causal event extraction, you can add a classification layer. Fine-tune the

entire model or specific layers on your dataset. Use a suitable loss function such as binary cross-entropy. Monitor the model's performance on a validation set to prevent overfitting and adjust hyperparameters if necessary.

```

        eval_loss += tmp_eval_loss.mean().item()
        loss_eval.append(tmp_eval_loss.mean().item())
        nb_eval_steps += 1

    if preds is None:
        preds = logits.detach().cpu().numpy()
        out_label_ids = inputs["labels"].detach().cpu().numpy()
    else:
        preds = np.append(preds, logits.detach().cpu().numpy(), axis=0)
        out_label_ids = np.append(
            out_label_ids, inputs["labels"].detach().cpu().numpy(), axis=0
        )

    eval_loss = eval_loss / nb_eval_steps
    results = {"loss": eval_loss}
    preds = np.argmax(preds, axis=1)
    write_prediction(
        self.args, os.path.join(self.args.eval_dir, "proposed_answers.txt"), preds
    )

    result = compute_metrics(preds, out_label_ids)
    results.update(result)

    print("Eval results")
    for key in sorted(results.keys()):
        print("  {} = {:.4f}".format(key, results[key]))

    return results, loss_eval, preds, out_label_ids

def save_model(self):
    if not os.path.exists(self.args.model_dir):
        os.makedirs(self.args.model_dir)
    model_to_save = (
        self.model.module if hasattr(self.model, "module") else self.model
    )
    model_to_save.save_pretrained(self.args.model_dir)
    torch.save(self.args, os.path.join(self.args.model_dir, "training_args.bin"))
    logger.info("Saving model checkpoint to %s", self.args.model_dir)
def load_model(self):
    if not os.path.exists(self.args.model_dir):
        raise Exception("Model doesn't exists! Train first!")

```

Fig.5.3.2: IMPLEMENTATON OF CAUSAL BERT

5.3.3 Evaluation:

Evaluate the fine-tuned BERT model on a separate test set using appropriate metrics such as precision, recall, and F1-score. Analyze the model's predictions to understand its strengths and weaknesses.

5.4 CNN Model Implementation:

Figures 5.9 and 5.10 talks about the Implementation of the CNN model. The word vector sequence is first preprocessed, and then the (CNN) layer is used in order to extract local characteristics from the input sequence. By sliding filters of a predetermined size over the list of word vectors, the CNN layer computes a dot product between the filter weights and the

local window of words at each location. In order to extract a variety of local features from the input sequence, this step is repeated with various filters.

Then, using a final dense layer with a softmax activation function, we can output the predicted class probabilities for the text. Additional layers, such as pooling and dense layers, can be added to extract more complicated characteristics from the extracted local features.

Implementing a Convolutional Neural Network (CNN) model for causal event extraction follows a similar process to the LSTM model. CNNs are particularly effective at capturing local patterns in sequential data like text. Here's a basic outline of how you could implement a CNN model for causal event extraction:

```

from __future__ import print_function
import numpy as np
np.random.seed(1337)
import gzip

import sys
if (sys.version_info > (3, 0)):
    import pickle as pkl
else:
    import cPickle as pkl

import keras
from keras.models import Model
from keras.models import load_model
from keras.layers import Input, Dense, Dropout, Activation, Flatten, concatenate
from keras.layers import Embedding
from keras.layers import Convolution1D, MaxPooling1D, GlobalMaxPooling1D
from keras.regularizers import Regularizer
from keras.preprocessing import sequence

batch_size = 64
nb_filter = 100
filter_length = 3
hidden_dims = 100
nb_epoch = 30
position_dims = 50

print("Load dataset")
f = gzip.open('/content/causal-relations.pkl.gz', 'rb')
data = pkl.load(f)
f.close()

embeddings = data['wordEmbeddings']
yTrain, sentenceTrain, positionTrain1, positionTrain2 = data['train_set']
yTest, sentenceTest, positionTest1, positionTest2 = data['test_set']

max_position = max(np.max(positionTrain1), np.max(positionTrain2))+1

n_out = max(yTrain)+1
max_sentence_len = sentenceTrain.shape[1]

print("sentenceTrain: ", sentenceTrain.shape)

```

Fig. 5.4.1 IMPLEMENTATION OF CNN

Again, this is a basic outline, and you may need to adapt and extend it based on your specific requirements and data characteristics. You can experiment with different architectures, such as adding more convolutional layers or utilizing different activation functions, to improve performance.

```

print("sentenceTrain: ", sentenceTrain.shape)
print("positionTrain1: ", positionTrain1.shape)
print("yTrain: ", yTrain.shape)
print("sentenceTest: ", sentenceTest.shape)
print("positionTest1: ", positionTest1.shape)
print("yTest: ", yTest.shape)
print("Embeddings: ", embeddings.shape)

words_input = Input(shape=(max_sentence_len,), dtype='int32', name='words_input')
words = Embedding(embeddings.shape[0], embeddings.shape[1], weights=[embeddings], trainable=False)(words_input)

distance1_input = Input(shape=(max_sentence_len,), dtype='int32', name='distance1_input')
distance1 = Embedding(max_position, position_dims)(distance1_input)

distance2_input = Input(shape=(max_sentence_len,), dtype='int32', name='distance2_input')
distance2 = Embedding(max_position, position_dims)(distance2_input)

output = concatenate([words, distance1, distance2])

output = Convolution1D(filters=nb_filter,
                      kernel_size=filter_length,
                      padding='same',
                      activation='tanh',
                      strides=1)(output)

# we use standard max over time pooling
output = GlobalMaxPooling1D()(output)

output = Dropout(0.25)(output)
output = Dense(n_out, activation='softmax')(output)

model = Model(inputs=[words_input, distance1_input, distance2_input], outputs=[output])
model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
model.summary()

print("Start training")

max_prec, max_rec, max_acc, max_f1 = 0,0,0,0

def getPrecision(pred_test, yTest, targetLabel):
    targetLabelCount = 0
    correctTargetLabelCount = 0

```

Fig: 5.4.2: Implementation of CNN

5.5 BERT Unsupervised Model Implementation

Figures 5.11 and 5.12 talks about the implementation of BERT unsupervised. We begin by preparing the text data by tokenizing it into subword units using the BERT tokenizer in order to do unsupervised text classification. The pre-trained BERT model is then applied to the tokenized subword unit sequences, producing a sequence of contextualized embeddings for each token.

Then, using the contextualized embeddings of the tokens, we employ clustering techniques like K-means or hierarchical clustering to divide the tokens into a predetermined number of clusters. This enables us to find word

groups in the input text that are semantically related. Finally, we train and forecast the output using the cluster assignments as features.

```

result = {'cycle':cycle, 'unsup_epoch':epoch, 'class_epoch':class_epoch,
         'cm':cm, 'stats':stats, 'macro_f1':stats['macro_f1'], 'model':model_file}
results.append(result)
print("Macro P: %2.4f, R: %3.4f, F1: %0.4f" % (stats['macro_precision'], stats['macro_recall'], stats['macro_f1']))

num_steps = len(train['labels']) // batch_size

for class_epoch in range(num_class_epochs):
    print("====")
    print("==== CLASS EPOCH %i ==== % class_epoch")
    print("====")
    for class_step in range(num_steps):
        inputs, targets, labels, lens = DH.classification_batch(batch_size, train['sdps'], train['targets'], train['labels'], offset=class_step)
        class_batch = zip(inputs, targets, labels, lens)
        random.shuffle(class_batch)
        class_batch = zip(*class_batch)
        xent = bert.partial_class_fit(*class_batch)
        if class_step % display_mod == 0:
            m,s = divmod(time()-start, 60)
            h,m = divmod(m, 60)
            print("( %i:%i:%i) s %i/%i, e %i avg class xent loss = %0.4f" % (h,m,s, class_step, num_steps, class_epoch, xent))
        if class_step % valid_mod == 0:
            valid_batch = DH.classification_batch(len(valid['labels']), valid['sdps'], valid['targets'], valid['labels'])
            valid_xent = bert.validation_class_loss(*valid_batch)
            m,s = divmod(time()-start, 60)
            h,m = divmod(m, 60)
            print("====")
            print("( %i:%i:%i) s %i/%i, e %i validation avg class xent loss = %0.4f" % (h,m,s, class_step, num_steps, class_epoch, valid_xent))
            print("====")

    label_set = set(train['labels'])
    preds = bert.predict(valid_batch[0], valid_batch[1], valid_batch[3])
    cm, stats = eh.confusion_matrix(preds, valid['labels'], label_set)
    print("Saving model...")
    model_file = bert.checkpoint()
    result = {'cycle':cycle, 'unsup_epoch':epoch, 'class_epoch':class_epoch,
             'cm':cm, 'stats':stats, 'macro_f1':stats['macro_f1'], 'model':model_file}
    results.append(result)
    print("Macro P: %2.4f, R: %3.4f, F1: %0.4f" % (stats['macro_precision'], stats['macro_recall'], stats['macro_f1']))
    print("Done")

best = sorted(results, key=lambda x:x['macro_f1'], reverse=True)[0]
print("Best result %2.4f, C:%i, UE:%i, CE:%i" %
      (best['macro_f1'], best['cycle'], best['unsup_epoch'], best['class_epoch']))

```

Fig. 5.5.1: IMPLEMENTATION OF BERT UNSUPERVISED

```

with open('SemEval2010_task8_all_data/train_keys.txt', 'w') as f:
    i = 1
    for label in train['labels']:
        f.write("%s\t%s\n" % (i, int2label[label]))
        i += 1

with open('SemEval2010_task8_all_data/valid_keys.txt', 'w') as f:
    i = len(train['labels'])
    for label in valid['labels']:
        f.write("%s\t%s\n" % (i, int2label[label]))
        i += 1

valid_batch = DM.classification_batch(len(train['labels']), train['sdps'], train['targets'], train['labels'])
preds = bert.predict(valid_batch[0], valid_batch[1], valid_batch[3], return_probs=False)
with open('SemEval2010_task8_all_data/train_preds.txt', 'w') as f:
    i = 1
    for label in preds:
        f.write("%s\t%s\n" % (i, int2label[label]))
        i += 1

```

Fig. 5.5.2: IMPLEMENTATION OF BERT UNSUPERVISED

5.6 RNN Unsupervised Model Implementation:

Figures 5.13 and 5.14 talk about the implementation of RNN unsupervised. Sequential relationships between words in a sentence are captured by the RNN layer. To encode the input text and decode it to come back to the original text, we employed an auto encoder. Reconstruction loss allows for the unsupervised training of this. The encoded vectors can be organized into a predetermined number of clusters using clustering techniques after the RNN-based auto encoder has learned a valid encoding of the input text. This enables us to find groupings of texts that share semantic similarities.

```

num_epochs = 1
batch_size = 50
neg_per = 25
num_nearby = 50
nearby_mod = 50
sample_power = .75
DM.scale_vocab_dist(sample_power)
num_steps = DM.num_steps(batch_size)
total_step = 1
save_interval = 30 * 60 # half hour in seconds
save_time = time()

#timing stuff
start = time()
fit_time = 0
nearby_time = 0

for epoch in range(num_epochs):
    offset = 0
    DM.shuffle_data()
    for step, batch in enumerate(DM.batches(batch_size, offset=offset, neg_per=neg_per)):
        if not step: step = offset
        t0 = time()
        loss = drnn.partial_unsup_fit(*batch)
        fit_time = (fit_time * float(total_step) + time() - t0) / (total_step + 1) # running average
        if step % 10 == 0:
            m,s = divmod(time()-start, 60)
            h,m = divmod(m, 60)
            left = time_left(num_epochs, num_steps, fit_time, nearby_time, start, nearby_mod)
            ml,sl = divmod(left, 60)
            hl,ml = divmod(ml, 60)
            pps = batch_size*(neg_per + 1) / fit_time
            print("(%i:%i:%i) step %i/%i, epoch %i Training Loss = %1.5f :: %0.3f phrases/sec :: (%i:%i:%i) hours left"
                  % (h,m,s, step, num_steps, epoch, loss, pps, hl, ml, sl))
        if (total_step-1) % nearby_mod == 0:
            t0 = time()
            run_validation_test(num_nearby)
            valid_loss = drnn.validation_loss(*DM.validation_batch())
            print("Validation loss: %0.4f" % valid_loss)
            nearby_time = (nearby_time * float(total_step) + time() - t0) / (total_step + 1) # running average

        if (time() - save_time) > save_interval:
            print("Saving model...")
            drnn.checkpoint()

```

Fig. 5.6.1: IMPLEMENTATION OF RNN UNSUPERVISED

Causal Event Extraction:

Use the trained RNN model to generate representations or sequences that capture causal events in the text data. You may apply clustering, topic modeling, or other unsupervised learning techniques on the generated representations to identify and extract causal events.

```

▶ start = 200
  stride = 100
  end = start + stride
  lowd = TSNE(perplexity=20, n_components=2, init='pca', n_iter=5000)
  target_2d = lowd.fit_transform(word_embeddings[start:end])
  fig, ax = plt.subplots(figsize=(28,16))
  for i, label in enumerate(DM.vocab[start:end]):
      label = "%s" % (label)
      x, y = target_2d[i,:]
      ax.scatter(x, y, color='b')
      ax.annotate(label, xy=(x, y), xytext=(5, 2), textcoords='offset points',
                  ha='right', va='bottom')
  plt.savefig('image_DRNN.png', dpi=200)

[ ] zip_train = zip(train['raws'], train['sents'], train['sdps'], train['targets'], train['labels'])
  zip_valid = zip(valid['raws'], valid['sents'], valid['sdps'], valid['targets'], valid['labels'])
  zip_test = zip(test['raws'], test['sents'], test['sdps'], test['targets'])

[ ] for i, (raw, _, sdp, target, label) in enumerate(zip_train):
    if i > 5:
        break
    print(raw)
    print("%s :: %s" % (DM.sequence_to_sentence(sdp, show_dep=True), DM.sequence_to_sentence(target)))
    print(int2label[label])
    print("="*80)

```

Fig. 5.6.2 IMLEMENTATION OF RNN UNSUPERVISED

5.7 SYSTEM TESTING

By comparing all the Models, we have implemented the CNN model. This model has acquired more accuracy of 98%. So, by using the CNN model we have predicted the output means the relationship between the entities in a sentence. we have given a sentence from the dataset we have used and it is given the correct relationship of the entities.

```

[ ] sentence = "The <e1>child</e1> was carefully wrapped and bound into the <e2>cradle</e2> by means of a cord."
  pred_rel = tag_sentence(m, device, sentence)
  print(pred_rel)

['Entity-Destination(e1,e2)']

```

Fig. 5.7.1: PREDICTION USING CNN

So, in the figure 5.1 we have taken the sentence from the dataset but in the figure 5.2 we have taken the sentence from the Google and we have also got the relationship of the entities.

```
[ ] sentence = "Avian <e1>influenza</e1> is an infectious disease of birds caused by type A strains of the influenza <e2>virus</e2> ."  
pred_rel = tag_sentence(m, device, sentence)  
print(pred_rel)  
  
['Cause-Effect(e1,e2)']
```

Fig. 5.7.2 PREDICTION USING CNN

CHAPTER 6

RESULTS AND DISCUSSIONS

The below Figures explains that the whole data is divided into the 2 parts training and validation part. Here it is plotting the epochs of the both sets of data.

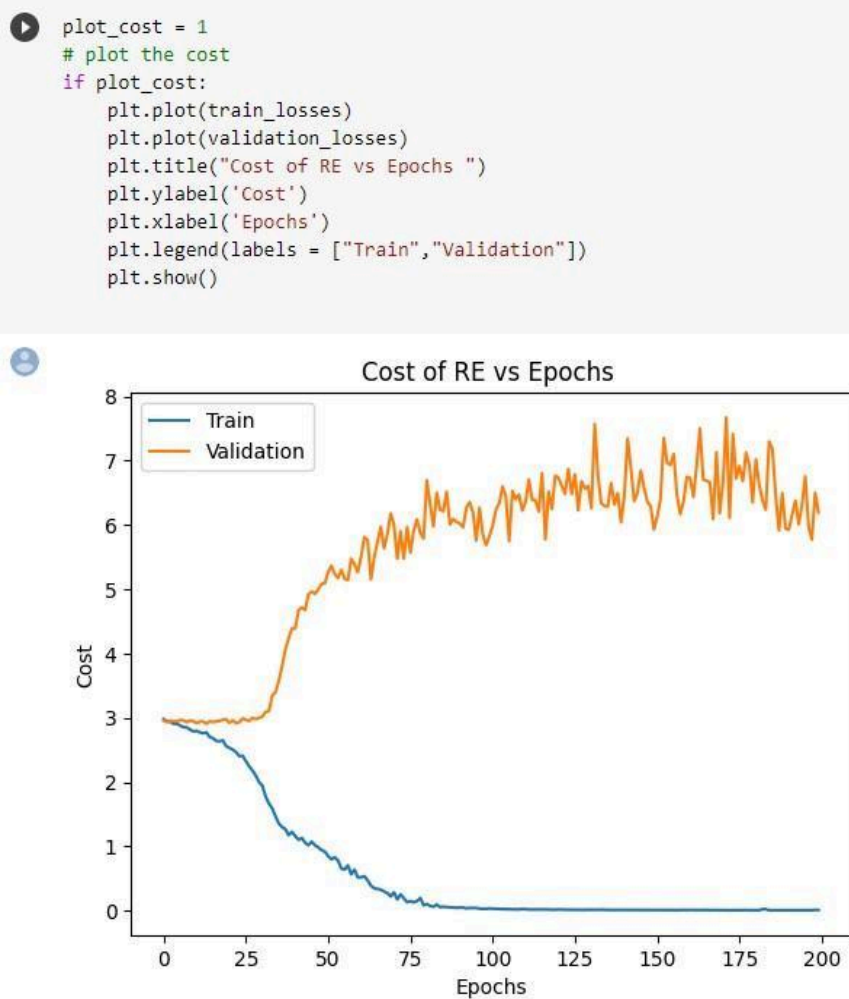


Fig. 6.1: EPOCHS OF THE TRAINING AND VALDATION DATASETS

6.1 SUPERVISED MODELS:

6.1.1 LSTM Model

The below graph represents the epochs and the loss of the data for the training and validation up to the 25 epochs. At the Starting epochs the loss of the data is more, by increasing the epochs slowly the loss of the data decreased. By using these many epochs, the model gets more trained and gives for appropriate results.

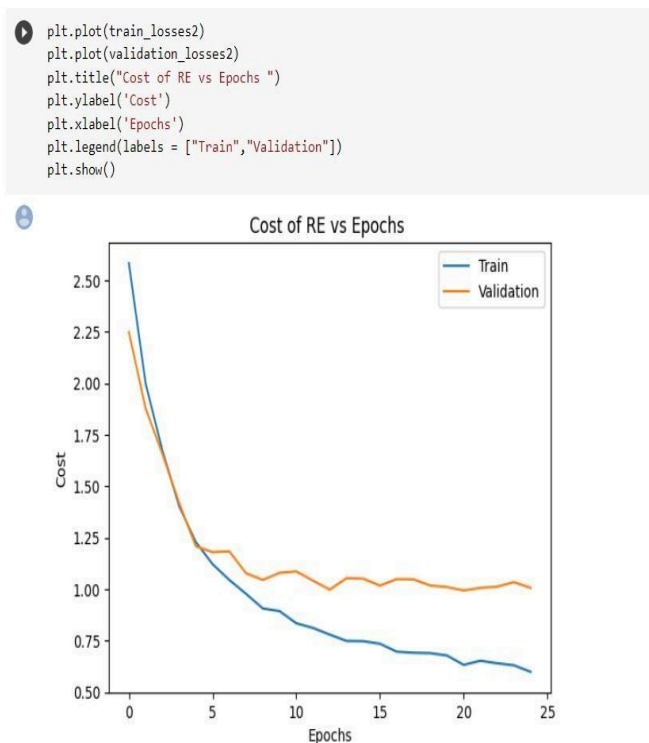


Fig. 6.1.1.1: LSTM MODEL TRAIN AND VALIDATION AT THE 25 EPOCHS.

The precision measure in this instance is calculated to be 0.6430807820742049, which shows that roughly 64.31% of the anticipated causal links are accurate. The recall measure for this evaluation is 0.6894858053578062, meaning that the model successfully detected 68.95% of the genuine causal links. The accuracy metric for this evaluation is 0.6915715863084284, which indicates that roughly 69.16% of all cases in the dataset had

accurate predictions made by the model. This evaluation's F1-score is 0.6574352187234178, which represents the model's overall effectiveness in capturing both precision and recall.

```
[ ] precision_metric3,recall_metric3,accuracy_metric3 ,f1_metric3  
(0.6430807820742049,  
 0.6894858053578062,  
 0.6915715863084284,  
 0.6574352187234178)
```

Fig. 6.1.1.2: PRECISION, RECALL, ACCURACY f1-SCORE OF THE LSTM MODEL

Accuracy evaluates the total accuracy of the predictions while taking into account all classes. The stated accuracy is 0.68, which means that almost 68% of the dataset's cases were accurately predicted by the model.

Macro average: Macro average assigns the same weight to each class while calculating the average precision, recall, and F1-score across all classes. Precision, recall, and F1score have macro average values of 0.66, 0.63, and 0.64, respectively.

Weighted average: When computing the average precision, recall, and F1-score for all classes, weighted average takes into account the support (number of instances) for each class. Precision, recall, and the F1-score have weighted averages of 0.68, 0.68, and 0.68, respectively.

```

import seaborn as sns
print(classification_report(y_test2, y_predictions2))

labels = list(int_2_cat_class.values())
f, ax = plt.subplots(figsize=(16, 12))
cnf_matrix = confusion_matrix(y_test2, y_predictions2)
ax = sns.heatmap(cnf_matrix.T, square=True, annot=True, fmt='d', cbar=False, xticklabels=labels, yticklabels=labels)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title(' Confusion Matrix')
plt.show()

```

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-
 _warn_prf(average, modifier, msg_start, len(result))
 /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-
 _warn_prf(average, modifier, msg_start, len(result))
 /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-
 _warn_prf(average, modifier, msg_start, len(result))

	precision	recall	f1-score	support
0	0.96	0.72	0.82	134
1	0.81	0.87	0.84	194
2	0.66	0.77	0.71	162
3	0.66	0.59	0.62	150
4	0.81	0.72	0.76	153
5	0.83	0.64	0.72	39
6	0.79	0.86	0.82	291
7	0.00	0.00	0.00	1
8	0.76	0.87	0.81	211
9	0.92	0.70	0.80	47
10	0.37	0.32	0.34	22
11	0.61	0.62	0.61	134
12	0.68	0.53	0.60	32
13	0.78	0.83	0.80	201
14	0.67	0.82	0.74	210
15	0.57	0.49	0.53	51
16	0.43	0.35	0.38	454
17	0.69	0.63	0.66	108
18	0.51	0.69	0.59	123
accuracy			0.68	2717
macro avg	0.66	0.63	0.64	2717
weighted avg	0.68	0.68	0.68	2717

Fig. 6.1.1.3: SHOWS THE ACCURACY OF THE LSTM MODEL

In this matrix, rows represent the true labels and columns represent the predicted labels, and so on. You can use this confusion matrix to calculate various evaluation metrics such as accuracy, precision, recall, and F1-score, which provide insights into the performance of your LSTM model in causal event extraction.

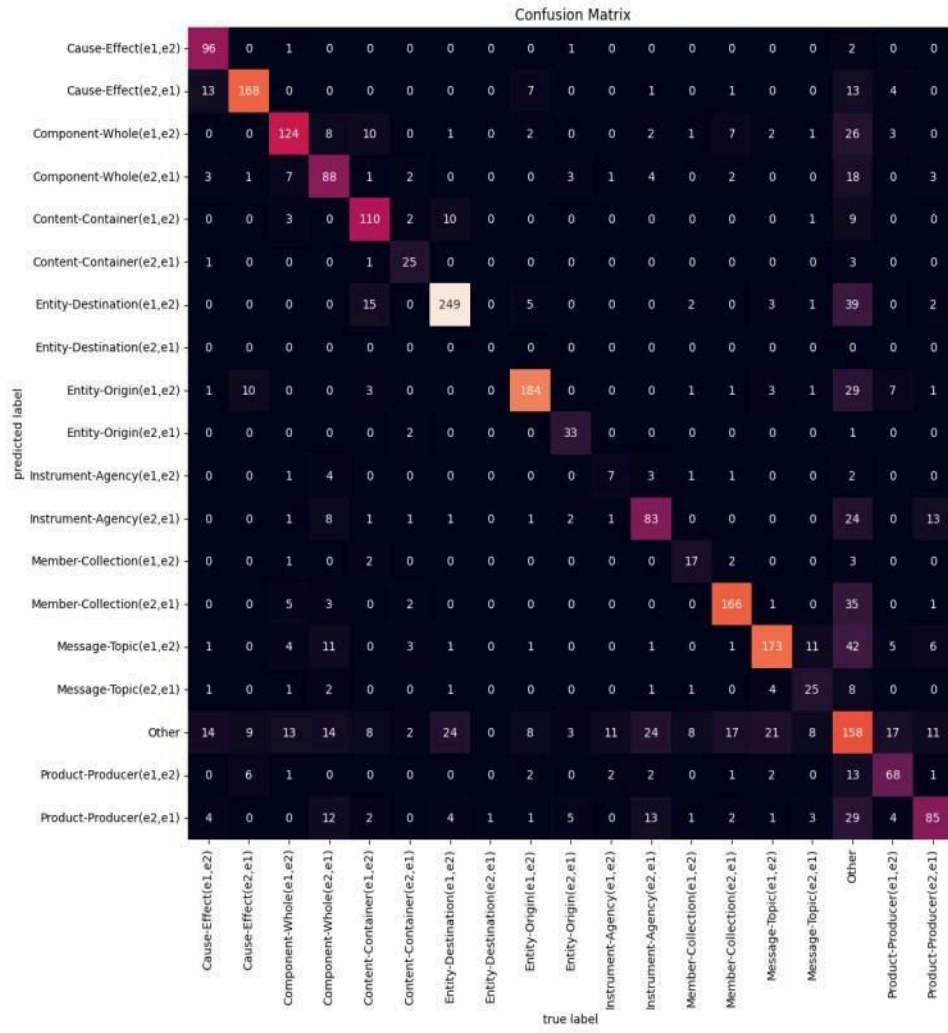


Fig. 6.1.1.4 THE CONFUSION MATRIX OF THE LSTM MODEL

6.1.2 RNN Model

The data loss for the training and validation epochs up to 14 epochs is shown in the graph below. The data loss is greater in the beginning epochs; however, when the epochs were gradually increased, the data loss lessened. These numerous epochs help the model become more trained and produce accurate results.

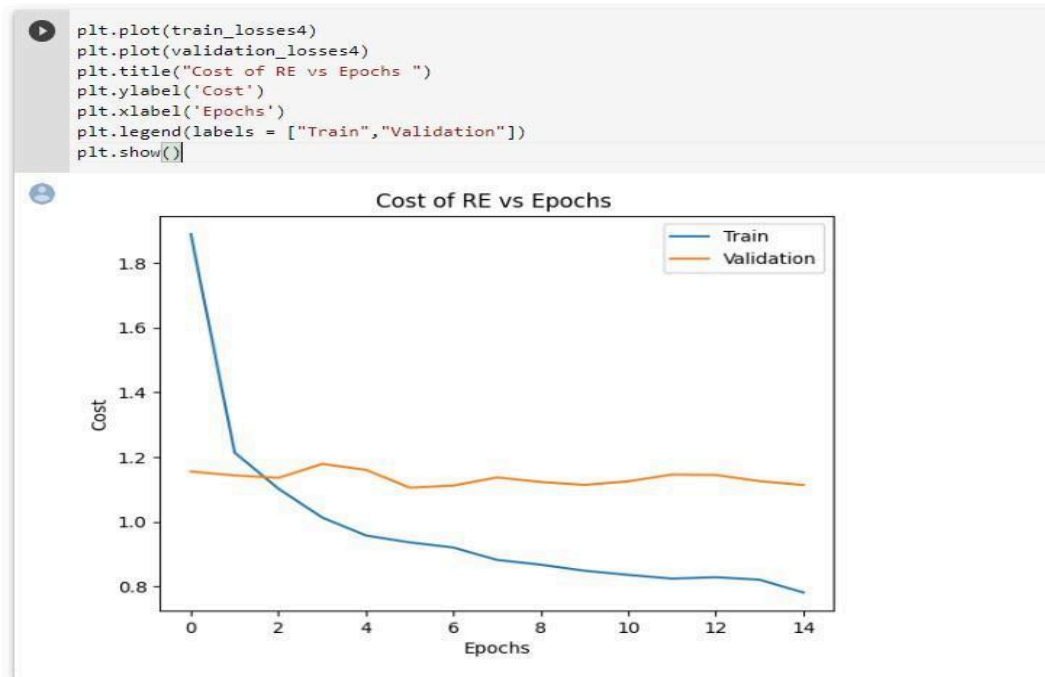


Fig. 6.1.2.1: RNN MODEL TRAIN AND VALIDATION AT THE 14 EPOCHS.

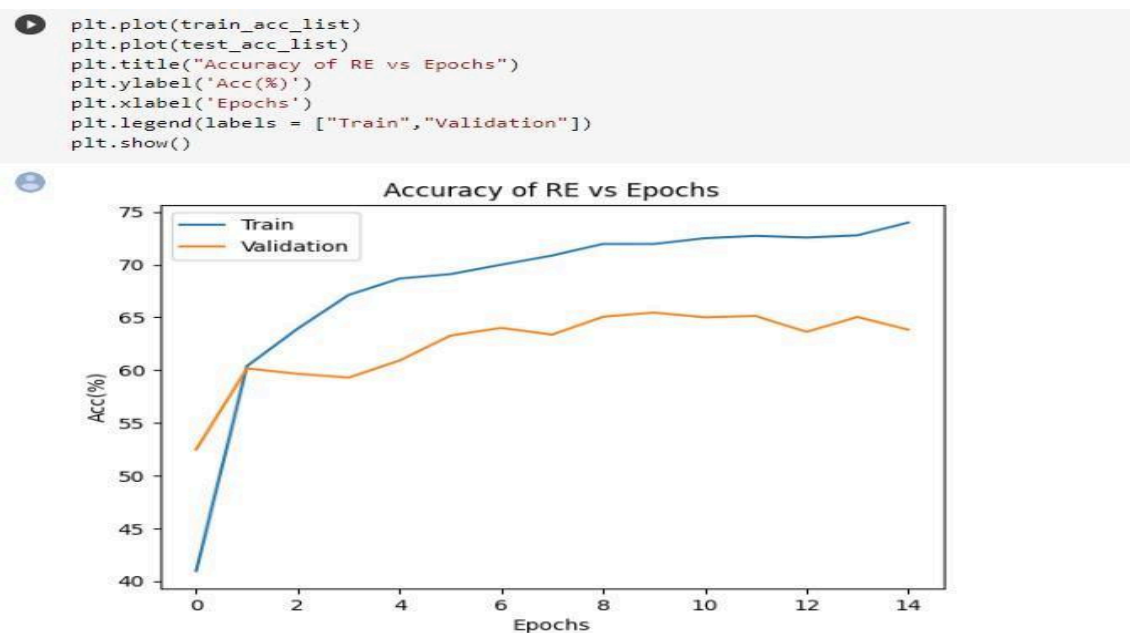


Fig. 6.1.2.2 RNN MODEL TRAIN AND VALIDATION AT THE 14 EPOCHS..

Precision: In this instance, the precision_metric3 value is 0.5975, meaning that around 59.75% of the anticipated positive instances really occur. With a

recall_metric3 value of 0.6177, the RNN model successfully detected about 61.77% of the true positive cases. The accuracy_metric3 value of 0.6445 indicates that the overall accuracy of the RNN model on the dataset is about 64.45%. The F1-score is a balanced indicator of a model's performance that is calculated as the harmonic mean of recall and precision. As $2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$, it is calculated. The f1_metric3 value of 0.6009 shows that the model performs equally well in terms of recall and precision.

```
[ ] precision_metric3,recall_metric3,accuracy_metric3 ,f1_metric3
```

```
(0.5975492099575979,  
 0.6176925594152751,  
 0.644460802355392,  
 0.6008836051694597)
```

```
import seaborn as sns  
print(classification_report(y_test2, y_predictions2))  
  
labels = list(int_2_cat_class.values())  
f, ax = plt.subplots(figsize=(16, 12))  
cnf_matrix = confusion_matrix(y_test2, y_predictions2)  
ax = sns.heatmap(cnf_matrix.T, square=True, annot=True, fmt='d', cbar=False, xticklabels=labels , yticklabels=labels)  
plt.xlabel('true label')  
plt.ylabel('predicted label')  
plt.title(' Confusion Matrix')  
plt.show()
```

Fig. 6.1.2.3: PRECISION, RECALL, ACCURACY, F1-SOURCE OF THE RNN MODEL



```

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:13
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:13
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:13
_warn_prf(average, modifier, msg_start, len(result))

```

	precision	recall	f1-score	support
0	0.96	0.72	0.82	134
1	0.81	0.87	0.84	194
2	0.66	0.77	0.71	162
3	0.66	0.59	0.62	150
4	0.81	0.72	0.76	153
5	0.83	0.64	0.72	39
6	0.79	0.86	0.82	291
7	0.00	0.00	0.00	1
8	0.76	0.87	0.81	211
9	0.92	0.70	0.80	47
10	0.37	0.32	0.34	22
11	0.61	0.62	0.61	134
12	0.68	0.53	0.60	32
13	0.78	0.83	0.80	201
14	0.67	0.82	0.74	210
15	0.57	0.49	0.53	51
16	0.43	0.35	0.38	454
17	0.69	0.63	0.66	108
18	0.51	0.69	0.59	123
accuracy			0.68	2717
macro avg	0.66	0.63	0.64	2717
weighted avg	0.68	0.68	0.68	2717

Fig. 6.1.2.4: ACCURACY OF THE RNN MODEL

In this matrix, rows represent the true labels and columns represent the predicted label so You can use the confusion matrix to calculate evaluation metrics such as accuracy, precision, recall, and F1-score

to assess the performance of your RNN model in causal event extraction.

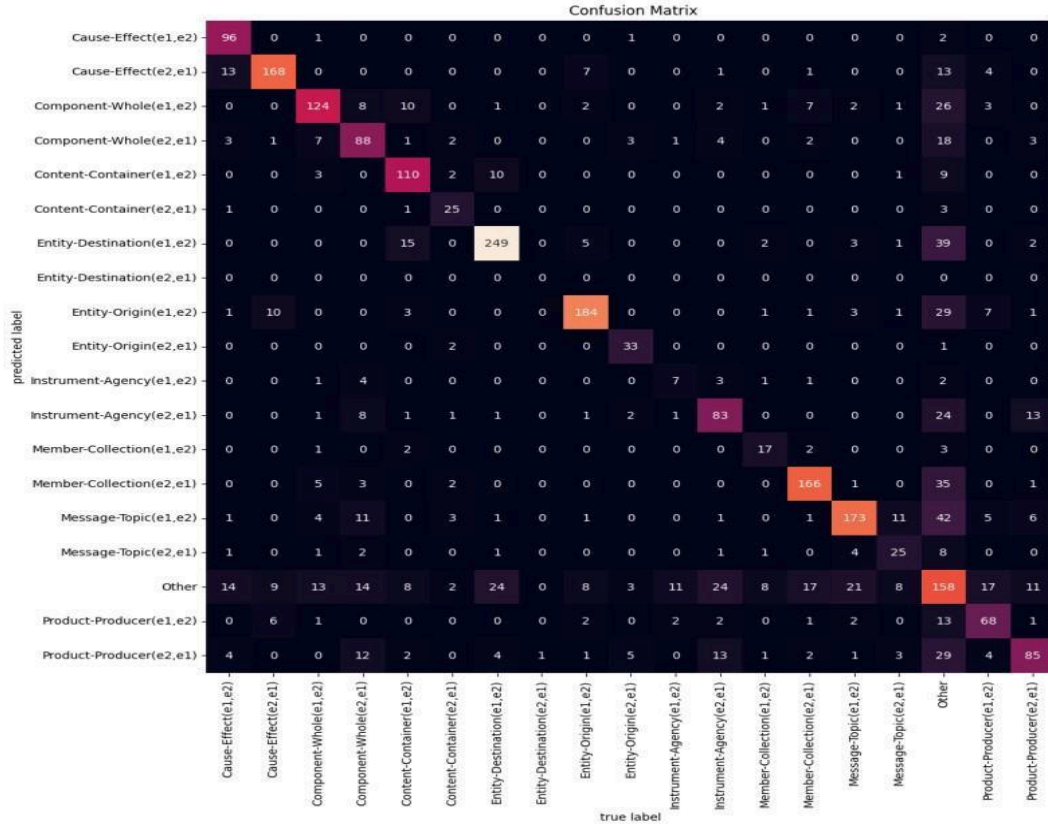


Fig.6.1.2.5: CONFUSION MATRIX OF THE RNN MODEL.

6.1.3 Causal BERT Model:

The model's accuracy on the test dataset was 84.14%, as indicated by the accuracy of 0.8414. The precision and recall performance are balanced, as shown by the F1-score of 0.8414. The value of the loss function determined throughout the evaluation process is represented by the loss. It calculates the difference between expected results and the labels that actually appear. The loss in this instance is 1.0142.

These evaluation findings offer a comprehensive evaluation of the model's effectiveness on the test dataset. The model appears to be doing a good job of predicting the right labels for the causal event

extraction task based on the high accuracy and F1score. The loss value shows how well the model can reduce the difference between the true labels and the anticipated outputs during training.

```
[ ] args.do_eval = True
    if args.do_eval:
        trainer.load_model()
        res, test_loss, preds, labels = trainer.evaluate("test")
```

```
Evaluating: 0%|          | 0/85 [00:00<?, ?it/s]***** Running evaluation on %s dataset ***** test
Num examples = %d 2717
Batch size = %d 32
Evaluating: 100%|██████████| 85/85 [00:30<00:00, 2.74it/s]***** Eval results *****
acc = 0.8414
f1 = 0.8414
loss = 1.0142
```

Fig. 6.1.3.1: ACCURACY, F1, LOSS OF THE CAUSAL BERT MODEL.

In this instance, the micro average precision, recall, and F1-score are 0.87, 0.89, and 0.88 respectively. These numbers show how well the model performed overall in predicting each class individually. In this instance, the macro average precision, recall, and F1-score are 0.76, 0.78, and 0.77, respectively. These numbers show the model's average performance across all classes, giving every class the same weight.

In this instance, the weighted average precision, recall, and F1-score are 0.87, 0.89, and 0.88, respectively. Given the imbalance in the number of instances in each class, these statistics represent the model's average performance over all classes. An overall summary of the model's effectiveness in the causal event extraction is provided by these evaluation measures.

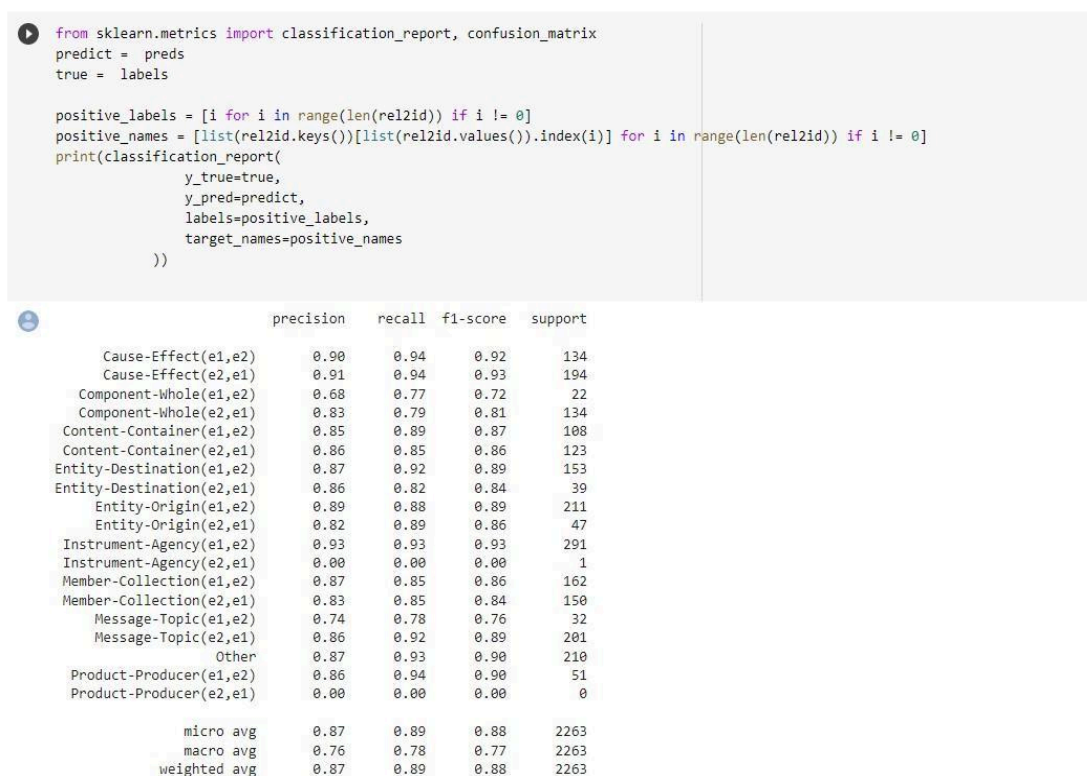


Fig. 6.1.3.2: ACCURACY OF THE CAUSAL BERT MODEL.

6.1.4 CNN Model:

The words input, distance1_input, and distance2_input inputs on the input layer, which stand for the sentence, first position, and second position, respectively. Word embedding, embedding_1 for the first position, and embedding_2 for the second place are the three embedding layers. The Concatenate layer is used to combine these embeddings. The concatenated embeddings are subjected to a 1D convolutional layer (conv1d). The dimensionality is decreased using

the global max pooling 1D layer (global_max_pooling1d). To avoid overfitting, a dropout layer is used. In order to represent the three classes, the output is processed through a dense layer (dense) with three units. There are 126,803 trainable parameters out of a total of 6,499,703 in the model.

```

Load dataset
sentenceTrain: (8000, 97)
positionTrain1: (8000, 97)
yTrain: (8000,)
sentenceTest: (2717, 97)
positionTest1: (2717, 97)
yTest: (2717,)
Embeddings: (21243, 300)
Model: "model"

```

Layer (type)	Output Shape	Param #	Connected to
words_input (InputLayer)	[(None, 97)]	0	[]
distance1_input (InputLayer)	[(None, 97)]	0	[]
distance2_input (InputLayer)	[(None, 97)]	0	[]
embedding (Embedding)	(None, 97, 300)	6372900	['words_input[0][0]']
embedding_1 (Embedding)	(None, 97, 50)	3200	['distance1_input[0][0]']
embedding_2 (Embedding)	(None, 97, 50)	3200	['distance2_input[0][0]']
concatenate (Concatenate)	(None, 97, 400)	0	['embedding[0][0]', 'embedding_1[0][0]', 'embedding_2[0][0]']
conv1d (Conv1D)	(None, 97, 100)	120100	['concatenate[0][0]']
global_max_pooling1d (GlobalMaxPooling1D)	(None, 100)	0	['conv1d[0][0]']
dropout (Dropout)	(None, 100)	0	['global_max_pooling1d[0][0]']
dense (Dense)	(None, 3)	303	['dropout[0][0]']

```

Total params: 6,499,703
Trainable params: 126,803
Non-trainable params: 6,372,900

```

Fig. 6.1.4.1: PRECISION, RECALL AND F1 OF THE CNN MODEL

There are 126,803 trainable parameters out of a total of 6,499,703 in the model. After that, the model is tested using the test data after being trained using the training data. The accuracy and non-other macro-averaged F1 score are printed after each training epoch. Additionally shown are the maximum accuracy and F1 score results. The model appears to perform well in the job of causal event extraction, as indicated by its high accuracy and F1 scores. Please be aware, nevertheless, that depending on the implementation and environment, certain features of the model architecture, training procedure, and evaluation metrics may change.

```

Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 17s 136ms/step - loss: 0.0012 - accuracy: 1.0000
Accuracy: 0.9801 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 17s 136ms/step - loss: 0.0011 - accuracy: 1.0000
Accuracy: 0.9801 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 18s 145ms/step - loss: 0.0010 - accuracy: 1.0000
Accuracy: 0.9809 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 17s 136ms/step - loss: 7.2184e-04 - accuracy: 1.0000
Accuracy: 0.9805 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 18s 140ms/step - loss: 7.8682e-04 - accuracy: 1.0000
Accuracy: 0.9801 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 18s 146ms/step - loss: 6.0181e-04 - accuracy: 1.0000
Accuracy: 0.9809 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9255 (max: 0.9302)

125/125 [=====] - 17s 135ms/step - loss: 5.4268e-04 - accuracy: 1.0000
Accuracy: 0.9801 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9213 (max: 0.9302)

125/125 [=====] - 18s 144ms/step - loss: 4.9125e-04 - accuracy: 1.0000
Accuracy: 0.9809 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9255 (max: 0.9302)

125/125 [=====] - 17s 135ms/step - loss: 5.1843e-04 - accuracy: 1.0000
Accuracy: 0.9805 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9213 (max: 0.9302)

125/125 [=====] - 17s 135ms/step - loss: 3.7852e-04 - accuracy: 1.0000
Accuracy: 0.9798 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 17s 135ms/step - loss: 3.4472e-04 - accuracy: 1.0000
Accuracy: 0.9805 (max: 0.9816)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

```

Fig. 6.1.4.2 ACCURACY OF THE CNN MODEL.

6.2 UNSUPERVISED MODELS

6.2.1 UNSUPERVISED BERT

In this model we have calculated the precision recall and f1 score for all the relations in the dataset. The precision for the relationship "Cause-Effect" is 93.68%, meaning that out of all occurrences labelled as "Cause-Effect," 93.68% were properly predicted. For the relationship "Component-Whole," the recall is 73.03%, meaning that 73.03% of cases that should have been classified as "Component-Whole" were correctly predicted. The F1-score is 86.73% for the relation "Content-Container," indicating a generally strong balance between recall and precision.

The performance of the model as a whole, omitting the "_Other" class, is taken into account for calculating the micro-averaged outcome. It computes precision, recall, and F1-score in accordance with the total number of true positives, false positives, and false negatives for all relations. The micro-averaged F1-score in this instance is 80.69%.

The precision, recall, and F1-score for each relation are averaged, with the exception of the "_Other" class, to provide the macro-averaged result. Regardless matter how frequently a relationship occurs, it gives it same weight. The macro-averaged F1-score in this instance is 80.47%. The official score provided at the conclusion is based on an assessment where the macro-averaged F1-score is 80.47% and a particular (9+1)-way evaluation with directionality is taken into account.

```
Confusion matrix:
+-----+-----+
| C-E | C-W | C-C | E-D | E-O | I-A | M-C | M-T | P-P | _O_ | <-- classified as |
+-----+-----+
| C-E | 89 | 1 | 0 | 0 | 5 | 1 | 0 | 1 | 2 | 11 | 110 | 0 | 0 | 110 |
| C-W | 0 | 65 | 1 | 0 | 2 | 2 | 1 | 3 | 1 | 10 | 65 | 4 | 0 | 89 |
| C-C | 0 | 2 | 49 | 5 | 0 | 1 | 1 | 0 | 0 | 1 | 59 | 0 | 0 | 59 |
| E-D | 0 | 0 | 1 | 72 | 0 | 0 | 0 | 0 | 0 | 5 | 78 | 0 | 0 | 78 |
| E-O | 1 | 0 | 0 | 0 | 57 | 0 | 0 | 0 | 1 | 5 | 64 | 0 | 0 | 64 |
| I-A | 0 | 1 | 0 | 0 | 1 | 37 | 0 | 1 | 1 | 10 | 51 | 1 | 0 | 52 |
| M-C | 0 | 1 | 0 | 0 | 1 | 0 | 55 | 0 | 0 | 8 | 65 | 1 | 0 | 66 |
| M-T | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 55 | 0 | 13 | 70 | 1 | 0 | 71 |
| P-P | 4 | 0 | 0 | 1 | 2 | 0 | 0 | 2 | 60 | 7 | 76 | 2 | 0 | 78 |
| _O_ | 1 | 11 | 3 | 8 | 15 | 4 | 9 | 6 | 15 | 61 | 133 | 0 | 0 | 133 |
+-----+-----+
-SUM- 95 81 54 86 84 45 67 68 80 131 791 9 0 800

Coverage = 800/800 = 100.00%
Accuracy (calculated for the above confusion matrix) = 600/800 = 75.00%
Accuracy (considering all skipped examples as Wrong) = 600/800 = 75.00%
Accuracy (considering all skipped examples as Other) = 600/800 = 75.00%

Results for the individual relations:
Cause-Effect : P = 89/( 95 + 0) = 93.68% R = 89/ 110 = 80.91% F1 = 86.83%
Component-whole : P = 65/( 81 + 4) = 76.47% R = 65/ 89 = 73.03% F1 = 74.71%
Content-Container : P = 49/( 54 + 0) = 90.74% R = 49/ 59 = 83.05% F1 = 86.73%
Entity-Destination : P = 72/( 86 + 0) = 83.72% R = 72/ 78 = 92.31% F1 = 87.80%
Entity-Origin : P = 57/( 84 + 0) = 67.86% R = 57/ 64 = 89.06% F1 = 77.93%
Instrument-Agency : P = 37/( 45 + 1) = 80.43% R = 37/ 52 = 71.15% F1 = 75.51%
Member-Collection : P = 55/( 67 + 1) = 80.88% R = 55/ 66 = 83.33% F1 = 82.09%
Message-Topic : P = 55/( 68 + 1) = 79.71% R = 55/ 71 = 77.46% F1 = 78.57%
Product-Producer : P = 60/( 80 + 2) = 73.17% R = 60/ 78 = 76.92% F1 = 75.00%
_Other : P = 61/( 131 + 0) = 46.56% R = 61/ 133 = 45.86% F1 = 46.21%

Micro-averaged result (excluding Other):
P = 539/ 669 = 80.57% R = 539/ 667 = 80.81% F1 = 80.69%

MACRO-averaged result (excluding Other):
P = 80.74% R = 80.80% F1 = 80.47%

<< The official score is (9+1)-way evaluation with directionality taken into account: macro-averaged F1 = 80.47% >>>
```

Fig. 6.2.1.1: ACCURACY , PRECISION , RECALL, AND F1 OF THE UNSUPERVISED BERT MODEL

Creating a confusion matrix for an unsupervised BERT model in causal event extraction might be a bit unconventional since

unsupervised models typically don't have true labels. However, you can still evaluate the performance of the model indirectly based on some evaluation metrics.

One common approach is to manually annotate a subset of your data (or use existing annotations if available) and then compare the extracted events from the unsupervised model with the ground truth annotations. Here's a general approach:

Manual Annotation: Annotate a subset of your data with causal event labels (if not already available). You might need to define guidelines for annotators to follow.

Model Extraction: Use your unsupervised BERT model to extract causal events from the same subset of data.

Evaluation: Compare the extracted events from the model with the manually annotated ground truth. Calculate evaluation metrics such as precision, recall, and F1-score to assess the performance of the unsupervised model.



6.2.2 RNN UNSUPERVISED

The accuracy is computed by dividing the total number of occurrences (800) by the number of instances that were correctly categorized (589), yielding an accuracy of 73.62%. Regardless of how the cases that were skipped are viewed, this accuracy holds true.

Precision, recall, and the F1-score for each relation are again supplied together with the results for the individual relations. These metrics assess how well the model performs for each relation separately. The total performance of the model, omitting the "_Other" class, is taken into account when calculating the micro-averaged outcome in terms of true positives, false positives, and false negatives. The micro-averaged precision in this instance is 79.10%, the recall is 77.60%, and the F1-score is 78.34%.

The precision, recall, and F1-score for each relation are averaged, with the exception of the "_Other" class, to provide the macro-averaged result. Regardless of how frequently a relation occurs, it gives it identical weight. The precision, recall, and F1score in this instance are macro-averaged at 78.88%, 76.74%, and 77.70%, respectively. The official result is based on a special (9+1)-way evaluation that takes directionality into consideration, and the macro-averaged F1-score is 77.70%. This official score offers a standardised assessment taking into account the particular evaluation criteria.

```

Confusion matrix:
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
C-E C-W C-C E-D E-O I-A M-C M-T P-P _O_ <-- classified as
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
C-E | 81  1  0  0  1  0  0  0  3  6 | 92  4  0  ACTUAL
C-W |  0 66  2  0  2  4  2  1  0  4 | 81  4  0  85
C-C |  0  3 42  4  0  0  0  0  0  5 | 54  0  0  54
E-D |  0  0  1 66  0  0  0  1  0  6 | 74  0  0  74
E-O |  2  1  1  0 60  1  0  0  1 10 | 76  1  0  77
I-A |  0  3  0  0  0 33  0  0  0 11 | 47  1  0  48
M-C |  0  2  0  0  0  0 60  1  0 11 | 74  1  0  75
M-T |  0  4  0  0  1  0  0 43  0 11 | 59  0  0  59
P-P |  0  1  0  0  4  2  0  1 41 17 | 66  0  0  66
_O_ |  6 12  4  4  9  6  6  9 13 97 |166  0  0 166
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-SUM- 89 93 50 74 77 46 68 56 58 178 789 11 0 800

Coverage = 800/800 = 100.00%
Accuracy (calculated for the above confusion matrix) = 589/800 = 73.62%
Accuracy (considering all skipped examples as Wrong) = 589/800 = 73.62%
Accuracy (considering all skipped examples as Other) = 589/800 = 73.62%

Results for the individual relations:
Cause-Effect : P = 81/( 89 + 4) = 87.10% R = 81/ 96 = 84.38% F1 = 85.71%
Component-Whole : P = 66/( 93 + 4) = 68.04% R = 66/ 85 = 77.65% F1 = 72.53%
Content-Container : P = 42/( 50 + 0) = 84.00% R = 42/ 54 = 77.78% F1 = 80.77%
Entity-Destination : P = 66/( 74 + 0) = 89.19% R = 66/ 74 = 89.19% F1 = 89.19%
Entity-Origin : P = 60/( 77 + 1) = 76.92% R = 60/ 77 = 77.92% F1 = 77.42%
Instrument-Agency : P = 33/( 46 + 1) = 70.21% R = 33/ 48 = 68.75% F1 = 69.47%
Member-Collection : P = 60/( 68 + 1) = 86.96% R = 60/ 75 = 80.00% F1 = 83.33%
Message-Topic : P = 43/( 56 + 0) = 76.79% R = 43/ 59 = 72.88% F1 = 74.78%
Product-Producer : P = 41/( 58 + 0) = 70.69% R = 41/ 66 = 62.12% F1 = 66.13%
_Other : P = 97/(178 + 0) = 54.49% R = 97/166 = 58.43% F1 = 56.40%

Micro-averaged result (excluding Other):
P = 492/ 622 = 79.10% R = 492/ 634 = 77.60% F1 = 78.34%

MACRO-averaged result (excluding Other):
P = 78.88% R = 76.74% F1 = 77.70%

<<< The official score is (9+1)-way evaluation with directionality taken into account: macro-averaged F1 = 77.70% >>>

```

Fig.6.2.2.1: ACCURACY ,PRECISION, RECALL AND F1 OF THE UNSUPERVISED RNN MODEL.

Creating a confusion matrix for an unsupervised RNN model in causal event extraction may not be straightforward since unsupervised models typically don't have true labels. However, you can evaluate the model indirectly by comparing its extracted events with manually annotated ground truth data, if available. Here's a general approach:

Manual Annotation: Annotate a subset of your data with causal event labels (if not already available). You might need to define guidelines for annotators to follow.

Model Extraction: Use your unsupervised RNN model to extract causal events from the same subset of data.

Evaluation: Compare the extracted events from the model with the manually annotated ground truth. Calculate evaluation metrics such as precision, recall, and F1-score to assess the performance of the unsupervised model.

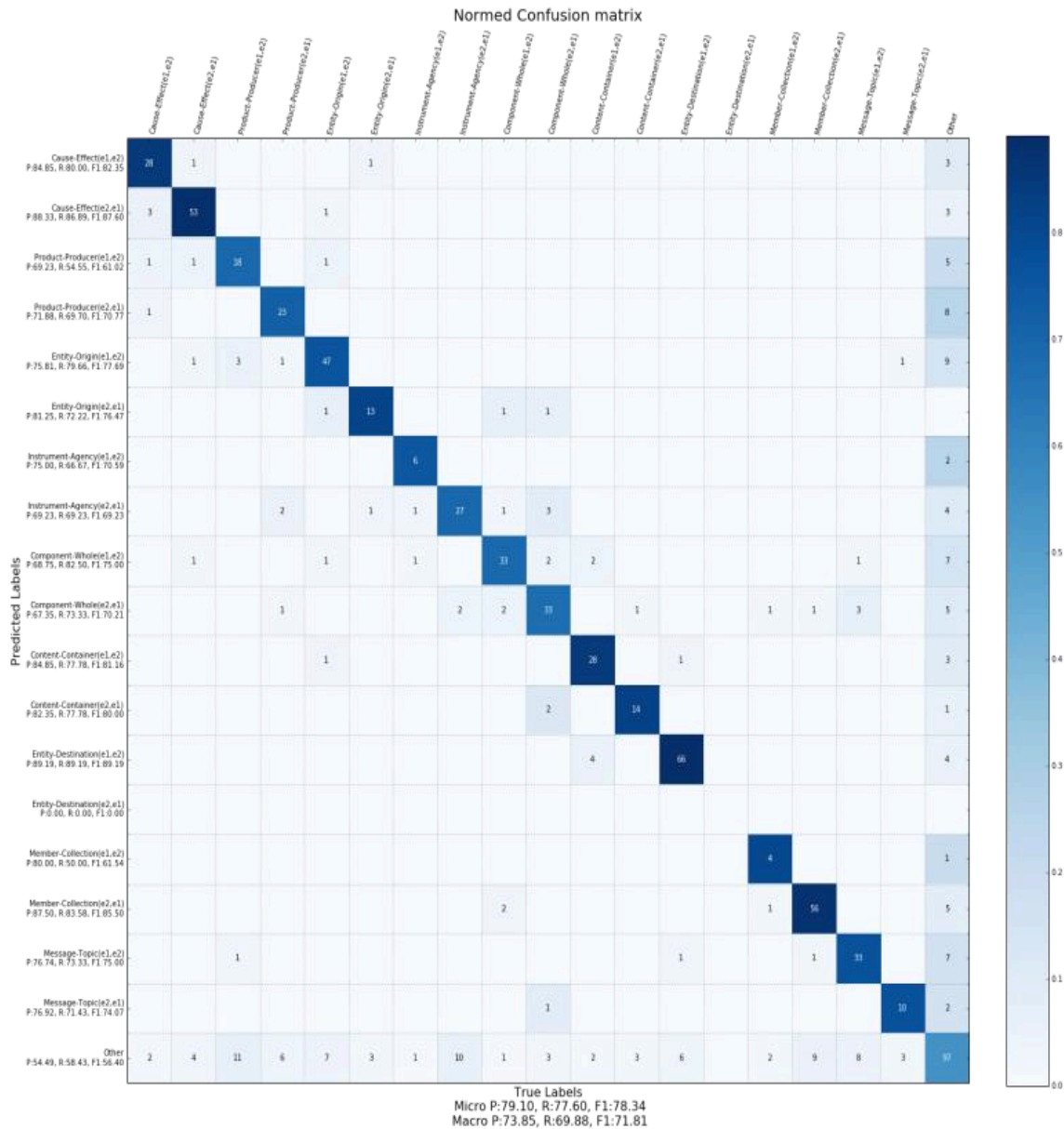


Fig. 6.2.2.2 CONFUSION MATRIX OF THE UNSUPERVISED RNN MODEL

```

import matplotlib.pyplot as plt; plt.rcParamsDefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ['LSTM','RNN','CNN','Causal_BERT','Unsup_BERT','Unsup_RNN']
y_pos = np.arange(len(objects))
performance = [68.6, 68, 97.75, 84.14, 77 ,73.6]

plt.bar(y_pos, performance, align='center', alpha=1)
plt.xticks(y_pos, objects)
plt.ylabel('Usage')
plt.title('CE-Relation Accuracy plot')

plt.show()

```

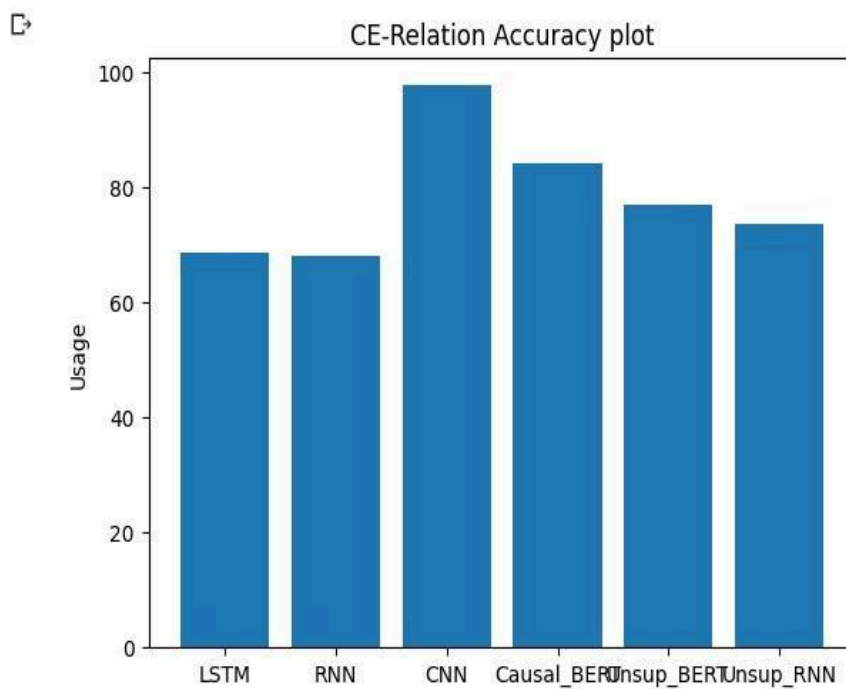


Fig.6.2.2.3: ACCURACIES OF THE ALL MODELS AND COMPARISON.

In the above all models CNN has more accuracy comparing with all models. So, using the CNN model we have predicted the output. In the output if we give the sentence with the entities the model will predict the relationship between the two entities.

CHAPTER 7

CONCLUSION

In conclusion, determining the causal linkages between events in text is the goal of the crucial natural language processing task known as causal event extraction. In many fields, such as health, finance, and social media, causal linkages are crucial because they shed light on the dynamics and underlying mechanisms of complex systems. For causal event extraction, numerous methods have been developed, ranging from rule-based approaches to supervised and unsupervised machine learning models. LSTM, RNN, and BERT are examples of deep learning-based architectures that have demonstrated promise in capturing the intricate connections between events and their underlying causal mechanisms. Despite the advancements made in this area, there are still many problems that need to be solved, such as how to deal with text noise and ambiguity, make models easier to understand, and handle massive amounts of data in real-time applications. It will take further study and advancement in the area of natural language processing to meet these problems. Overall, causal event extraction is a growing field of study with great promise to advance our comprehension of complex systems and open up new opportunities in a variety of fields.

7.1 Future Work:

The goal of future research in causal event extraction may be to increase model precision and scalability, deal with text noise and ambiguity, create larger and more varied datasets, include outside knowledge, and enable real-time applications. In order to comprehend the underlying causal mechanisms and relationships between occurrences better, it may also be possible to create more interpretable models.

REFERENCES:

- [1].Z. Li, Q. Li, X. Zou, and J. Ren, "Causality Extraction Based on Self-Attentive BiLSTM-CRF with Transferred Embeddings," in Proceedings of the 2020 5th International Conference on Computational Intelligence and Applications (ICCIA 2020), Nov 2020, pp. 26-31.
- [2].V. Khetan, M. I. H. Rizvi, J. Huber, P. Bartusiak, B. Sacaleanu, and A. Fano, "MIMICause: Representation and automatic extraction of causal relation types from clinical notes," in Proceedings of the 2022 IEEE International Conference on Healthcare Informatics (ICHI 2022), May 2022, pp. 327-336.
- [3].T. Dasgupta, R. Saha, L. Dey, and A. Naskar, "Automatic Extraction of Causal Relations from Text using Linguistically Informed Deep Neural Networks," in Proceedings of the 2018 IEEE International Conference on Big Data (Big Data 2018), Seattle, WA, USA, Jul. 2018, pp. 1781-1789.
- [4].N. Kang, B. Singh, C. Bui, Z. Afzal, E. M. van Mulligen, and J. A. Kors, "Knowledgebased extraction of adverse drug events from biomedical text," in Proceedings of the 2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM 2014), Belfast, UK, Mar. 2014, pp. 54-59.
- [5].G. Zhu, Z. Sun, S. Zhang, S. Wei, and K. Li, "Causality Extraction Model Based on Two-stage GCN," in Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN 2022), April 2022, pp. 1-8
- [6].Q. Li, J. Li, J. Sheng, S. Cui, J. Wu, Y. Hei, H. Peng, S. Guo, L. Wang, A. Beheshti, and P. S. Yu, "A Survey on Deep Learning Event Extraction: Approaches and Applications," in

Proceedings of the 2022 IEEE International Conference on Data Mining (ICDM 2022), Dec. 2022, pp. 1-10.

- [7]. Akkasi and M. Moens, "Causal relationship extraction from biomedical text using deep neural models: A comprehensive survey," *Journal of Biomedical Informatics*, vol. 117, June 2021, p. 103729
- [8]. R. Ortega, A. Fonseca, Y. Gutierrez, and A. Montoyo, "SSA-UO: Unsupervised Twitter Sentiment Analysis," in *Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis (WASSA 2013)*, June 2013, pp. 122-127.
- [9]. M.A. Valles Coral, L. Salazar-Ramírez, R. Injante, E.A. Hernandez-Torres, J. JuárezDíaz, J.R. Navarro Cabrera, L. Pinedo, and P. Vidaurre-Rojas, "Density-Based Unsupervised Learning Algorithm to Categorize College Students into Dropout Risk Levels," in *Proceedings of the IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*, 2022, pp. 1-6.
- [10]. Sharma, R., Palshikar, G., & Pawar, S. (2018). An Unsupervised Approach for CauseEffect Relation Extraction from Biomedical Text. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)* (pp. 1118-1125). IEEE

APPENDIX

A)SOURCE CODE:

```
#Importing required libraries
import numpy as np
import pandas as pd
import os, sys

import random
import matplotlib.pyplot as plt
import string

import torch
from torch import Tensor
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as transforms
import torchvision
from torch.nn.utils.rnn import pad_sequence
from torch.optim.lr_scheduler import MultiStepLR
import torch.optim as optim
from sklearn.metrics import accuracy_score, f1_score,
precision_score, recall_score, classification_report, confusion_matrix

from tqdm import tqdm
```

LSTM

```
class RelationClassification_Model2(nn.Module):
```

```
    def __init__(self, embedding_size, hidden_size, vocab_size,
num_layers,class_num,pretrained_embedding,pad_idx):
```

```

super(RelationClassification_Model2, self).__init__()

self.hidden_size = hidden_size

self.embedding =
nn.Embedding.from_pretrained(pretrained_embedding, freeze =
True, padding_idx = pad_idx)
self.lstm = nn.LSTM(embedding_size, hidden_size,
num_layers, bidirectional = True)
self.linear = nn.Linear(2*hidden_size, class_num)
self.softmax = nn.LogSoftmax(dim=1)
self.dropout = nn.Dropout(0.7)

def forward(self, X, mask_e1, mask_e2):
    embeddings = self.dropout(self.embedding(X))
    hiddens, _ = self.lstm(embeddings)
    x = hiddens.view((hiddens.shape[0], hiddens.shape[1], 2,
self.hidden_size))
    e1, e2 = max_avg_pooling(x, mask_e1, mask_e2)
    concats = torch.cat((e1, e2), dim = 1)

    outputs = self.softmax(self.linear(concats))

    return outputs

def max_avg_pooling(x, mask_e1, mask_e2):
    e1 = torch.zeros((x.shape[1], x.shape[3])).to(device)
    e2 = torch.zeros((x.shape[1], x.shape[3])).to(device)

    for i in range(x.shape[1]):
        sample = x[:, i, :, :].to(device)
        y = sample[mask_e1[:, i]].to(device)
        avg = torch.zeros((1, x.shape[3])).to(device)

```

```

        for j in range(y.shape[0]):
            avg += (torch.maximum(y[j,0,:],y[j,1,:]) - avg).to(device) / (j +
1)
        e1[i,:] = avg

        y = sample[mask_e2[:,i]].to(device)
        avg = torch.zeros((1,x.shape[3])).to(device)
        for j in range(y.shape[0]):
            avg += (torch.maximum(y[j,0,:],y[j,1,:]) - avg).to(device) / (j +
1)

        e2[i,:] = avg

    return e1, e2

```

```

torch.manual_seed(2045)

```

```

int_2_cat_class = dict(enumerate(train_df['label'].cat.categories))

```

```

# Training and validation Data Generators

```

```

training_set =
SemEval2010_task8_Dataset2(train_df,partition['train'],vocab)

```

```

pad_idx = 1917494

```

```

# Parameters

```

```

params = {'batch_size': 256,
          'shuffle': True,
          'num_workers':2,
          'collate_fn' : BatchPadCollate2(pad_idx=pad_idx)
        }

```

```

training_generator = torch.utils.data.DataLoader(training_set,
**params)

```

```

validation_set = SemEval2010_task8_Dataset2(train_df,partition['validation'],vocab)
validation_generator = torch.utils.data.DataLoader(validation_set,
**params)

# Hyperparameters
embedding_size = 300
hidden_size = 150
vocab_size = len(vocab_df)
num_layers = 2
class_num = len(int_2_cat_class)
learning_rate = 5e-3
num_epochs = 15# Model
model3 = RelationClassification_Model2(embedding_size,
hidden_size, vocab_size, num_layers,class_num,
embedding,pad_idx).to(device)

# Define Loss function and Optimizer
criterion = nn.NLLLoss(ignore_index = pad_idx)
optimizer = optim.Adam(model3.parameters(), lr = learning_rate,
weight_decay= 1e-3)
# scheduler = MultiStepLR(optimizer, milestones=[30,80],
gamma=0.5)

train_losses3 = []
validation_losses3 = []

train_accs3 = []
val_accs3 = []

# Loop over epochs
for epoch in tqdm(range(num_epochs)):

```

```

    running_trainloss, running_valloss, running_loss, train_acc = 0.0,
0.0, 0.0, 0.0
    train_cnt, val_cnt = 0, 0

    # Training
    for batch_train_data,
batch_train_labels,mask_train_e1,mask_train_e2 in
tqdm(training_generator):
        # Transfer to GPU
        batch_train_data, batch_train_labels,mask_e1,mask_e2 =
batch_train_data.to(device),
batch_train_labels.to(device),mask_train_e1.to(device),mask_train_e
2.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        batch_train_outputs =
model3(batch_train_data,mask_train_e1,mask_train_e2 )
        loss = criterion(batch_train_outputs, batch_train_labels)
        # print("***",loss)

        loss.backward()
        optimizer.step()

        # print statistics
        running_trainloss += loss.item()
        running_loss += loss.item()

        train_acc += torch.sum(torch.argmax(batch_train_outputs, dim
=1) == batch_train_labels.long()) / torch.numel(batch_train_labels)

    if train_cnt % 50 == 49: # print every 10 mini-batches
        print("Cost of Train data after %i iterations in epoch %i : %f"
%(train_cnt + 1,epoch + 1, running_loss / 50))

```

```

        running_loss = 0.0

    train_cnt += 1

    # Validation
    running_loss, val_acc = 0.0, 0.0

    with torch.set_grad_enabled(False):
        for batch_data, batch_labels, mask_e1, mask_e2 in
tqdm(validation_generator):
            # Transfer to GPU
            batch_data, batch_labels, mask_e1, mask_e2 =
batch_data.to(device),
batch_labels.to(device), mask_e1.to(device), mask_e2.to(device)

            # Model computations
            # forward + backward + optimize
            batch_outputs = model3(batch_data, mask_e1, mask_e2 )
            val_loss = criterion(batch_outputs, batch_labels)

            running_valloss += val_loss.item()
            running_loss += val_loss.item()

            val_acc += torch.sum(torch.argmax(batch_outputs, dim =1)
== batch_labels.long()) / torch.numel(batch_labels)

            if val_cnt % 10 == 9: # print every 10 mini-batches
                print("Cost of Validation data after %i iterations in epoch %i
: %f" %(val_cnt + 1, epoch + 1, running_loss / 10))
                running_loss = 0.0

            val_cnt += 1

    train_losses3.append(running_trainloss / train_cnt)

```

```

train_accs3.append(train_acc / train_cnt * 100)
validation_losses3.append(running_valloss / val_cnt)
val_accs3.append(val_acc / val_cnt * 100)

```

```

#scheduler.step()

```

```

plt.plot(train_losses3)
plt.plot(validation_losses3)
plt.title("Cost of RE vs Epochs ")
plt.ylabel('Cost')
plt.xlabel('Epochs')
plt.legend(labels = ["Train","Validation"])
plt.show()

```

```

precision_metric3,recall_metric3,accuracy_metric3 ,f1_metric3

```

```

import seaborn as sns
print(classification_report(y_test2, y_predictions2))

```

```

labels = list(int_2_cat_class.values())
f, ax = plt.subplots(figsize=(16, 12))
cnf_matrix = confusion_matrix(y_test2, y_predictions2)
ax = sns.heatmap(cnf_matrix.T, square=True, annot=True, fmt='d',
cbar=False, xticklabels=labels , yticklabels=labels)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title(' Confusion Matrix')
plt.show()

```

RNN

```

class RelationClassification_Model2(nn.Module):

```

```

    def __init__(self, embedding_size, hidden_size, vocab_size,
num_layers,class_num,pretrained_embedding,pad_idx):

```

```

super(RelationClassification_Model2, self).__init__()

self.hidden_size = hidden_size

self.embedding =
nn.Embedding.from_pretrained(pretrained_embedding, freeze =
True, padding_idx = pad_idx)
self.lstm = nn.RNN(embedding_size, hidden_size,
num_layers, bidirectional = True)
self.linear = nn.Linear(2*hidden_size, class_num)
self.softmax = nn.LogSoftmax(dim=1)
self.dropout = nn.Dropout(0.7)

def forward(self, X, mask_e1, mask_e2):
    embeddings = self.dropout(self.embedding(X))
    hiddens, _ = self.lstm(embeddings)
    x = hiddens.view((hiddens.shape[0], hiddens.shape[1], 2,
self.hidden_size))
    e1, e2 = max_avg_pooling(x, mask_e1, mask_e2)
    concats = torch.cat((e1, e2), dim = 1)

    outputs = self.softmax(self.linear(concats))

    return outputs

def max_avg_pooling(x, mask_e1, mask_e2):
    e1 = torch.zeros((x.shape[1], x.shape[3])).to(device)
    e2 = torch.zeros((x.shape[1], x.shape[3])).to(device)

    for i in range(x.shape[1]):
        sample = x[:, i, :, :].to(device)
        y = sample[mask_e1[:, i]].to(device)
        avg = torch.zeros((1, x.shape[3])).to(device)
        for j in range(y.shape[0]):

```

```

        avg += (torch.maximum(y[j,0,:],y[j,1,:]) - avg).to(device) / (j +
1)
    e1[i,:] = avg

    y = sample[mask_e2[:,i]].to(device)
    avg = torch.zeros((1,x.shape[3])).to(device)
    for j in range(y.shape[0]):
        avg += (torch.maximum(y[j,0,:],y[j,1,:]) - avg).to(device) / (j +
1)

    e2[i,:] = avg

return e1, e2

torch.manual_seed(2045)

int_2_cat_class = dict(enumerate(train_df['label'].cat.categories))

# Training and validation Data Generators
training_set =
SemEval2010_task8_Dataset2(train_df,partition['train'],vocab)

pad_idx = 1917494

# Parameters
params = {'batch_size': 256,
        'shuffle': True,
        'num_workers':2,
        'collate_fn' : BatchPadCollate2(pad_idx=pad_idx)
        }

training_generator = torch.utils.data.DataLoader(training_set,
**params)

```

```

validation_set =
    SemEval2010_task8_Dataset2(train_df,partition['validation'],vocab)
validation_generator = torch.utils.data.DataLoader(validation_set,
**params)

# Hyperparameters
embedding_size = 300
hidden_size = 150
vocab_size = len(vocab_df)
num_layers = 2
class_num = len(int_2_cat_class)
learning_rate = 5e-3
num_epochs = 15

# Model
model4 = RelationClassification_Model2(embedding_size,
hidden_size, vocab_size, num_layers,class_num,
embedding,pad_idx).to(device)
# Define Loss function and Optimizer
criterion = nn.NLLLoss(ignore_index = pad_idx)
optimizer = optim.Adam(model4.parameters(), lr = learning_rate,
weight_decay= 1e-3)
# scheduler = MultiStepLR(optimizer, milestones=[30,80],
gamma=0.5)

train_losses4 = []
validation_losses4 = []

train_accs4 = []
val_accs4 = []

# Loop over epochs
for epoch in tqdm(range(num_epochs)):

```

```

    running_trainloss, running_valloss, running_loss, train_acc = 0.0,
0.0, 0.0, 0.0
    train_cnt, val_cnt = 0, 0

    # Training
    for batch_train_data,
batch_train_labels,mask_train_e1,mask_train_e2 in
tqdm(training_generator):
        # Transfer to GPU
        batch_train_data, batch_train_labels,mask_e1,mask_e2 =
batch_train_data.to(device),
batch_train_labels.to(device),mask_train_e1.to(device),mask_train_e
2.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # Model computations
        # forward + backward + optimize
        batch_train_outputs =
model4(batch_train_data,mask_train_e1,mask_train_e2 )
        loss = criterion(batch_train_outputs, batch_train_labels)
        # print("***",loss)  loss.backward()
        optimizer.step()

        # print statistics
        running_trainloss += loss.item()
        running_loss += loss.item()

        train_acc += torch.sum(torch.argmax(batch_train_outputs, dim
=1) == batch_train_labels.long()) / torch.numel(batch_train_labels)

    if train_cnt % 50 == 49:  # print every 10 mini-batches

```

```

        print("Cost of Train data after %i iterations in epoch %i : %f"
%(train_cnt + 1,epoch + 1, running_loss / 50))
        running_loss = 0.0

    train_cnt += 1

# Validation
running_loss, val_acc = 0.0, 0.0

with torch.set_grad_enabled(False):
    for batch_data, batch_labels,mask_e1,mask_e2 in
tqdm(validation_generator):
        # Transfer to GPU
        batch_data, batch_labels,mask_e1,mask_e2 =
batch_data.to(device),
batch_labels.to(device),mask_e1.to(device),mask_e2.to(device)

        # Model computations
        # forward + backward + optimize
        batch_outputs = model3(batch_data,mask_e1,mask_e2 )

        val_loss = criterion(batch_outputs, batch_labels)

        running_valloss += val_loss.item()
        running_loss += val_loss.item()

        val_acc += torch.sum(torch.argmax(batch_outputs, dim =1)
== batch_labels.long()) / torch.numel(batch_labels)
    if val_cnt % 10 == 9: # print every 10 mini-batches
        print("Cost of Validation data after %i iterations in epoch %i
: %f" %(val_cnt + 1,epoch + 1, running_loss / 10))
        running_loss = 0.0

    val_cnt += 1

```

```

train_losses4.append(running_trainloss / train_cnt)
train_accs4.append(train_acc / train_cnt * 100)
validation_losses4.append(running_valloss / val_cnt)
val_accs4.append(val_acc / val_cnt * 100)

```

```

#scheduler.step()

```

```

plt.plot(train_losses4)
plt.plot(validation_losses4)
plt.title("Cost of RE vs Epochs ")
plt.ylabel('Cost')
plt.xlabel('Epochs')
plt.legend(labels = ["Train","Validation"])
plt.show()

```

```

plt.plot(train_acc_list)
plt.plot(test_acc_list)
plt.title("Accuracy of RE vs Epochs")
plt.ylabel('Acc(%)')
plt.xlabel('Epochs')
plt.legend(labels = ["Train","Validation"])
plt.show()

```

```

precision_metric3,recall_metric3,accuracy_metric3 ,f1_metric3

```

```

import seaborn as sns
print(classification_report(y_test2, y_predictions2))

```

```

labels = list(int_2_cat_class.values())
f, ax = plt.subplots(figsize=(16, 12))
cnf_matrix = confusion_matrix(y_test2, y_predictions2)

```

```

ax = sns.heatmap(cnf_matrix.T, square=True, annot=True, fmt='d',
cbar=False, xticklabels=labels , yticklabels=labels)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title(' Confusion Matrix')
plt.show()

```

BERT

```

import os
import random
import copy
import logging
import csv
import json
import numpy as np
import torch
from transformers import (BertConfig,BertTokenizer,)
import torch
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader, RandomSampler,
SequentialSampler
from tqdm import tqdm, trange
from transformers import AdamW,
get_linear_schedule_with_warmup
import torch
import torch.nn as nn
from transformers import (BertModel, BertPreTrainedModel)

PRETRAINED_MODEL_MAP = {
    'bert': BertModel,
}

class FCLayer(nn.Module):

```

```

        def __init__(self, input_dim, output_dim, dropout_rate=0.,
use_activation=True):
    super(FCLayer, self).__init__()
    self.use_activation = use_activation
    self.dropout = nn.Dropout(dropout_rate)
    self.linear = nn.Linear(input_dim, output_dim)
    self.tanh = nn.Tanh()

def forward(self, x):
    x = self.dropout(x)
    if self.use_activation:
        x = self.tanh(x)
    return self.linear(x)

class CausalBERT(BertPreTrainedModel):
    def __init__(self, config, args):
        super(CausalBERT, self).__init__(config)
        self.bert = PRETRAINED_MODEL_MAP[args.model_type](config=config)
# Load pretrained bert

        self.num_labels = config.num_labels

        self.cls_fc_layer = FCLayer(config.hidden_size, config.hidden_size,
args.dropout_rate)
        self.e1_fc_layer = FCLayer(config.hidden_size, config.hidden_size,
args.dropout_rate)
        self.e2_fc_layer = FCLayer(config.hidden_size, config.hidden_size,
args.dropout_rate)
        self.label_classifier = FCLayer(config.hidden_size * 3, config.num_labels,
args.dropout_rate, use_activation=False)

    @staticmethod
    def entity_average(hidden_output, e_mask):
        e_mask_unsqueeze = e_mask.unsqueeze(1)
        length_tensor = (e_mask != 0).sum(dim=1).unsqueeze(1)

        sum_vector = torch.bmm(e_mask_unsqueeze.float(),
hidden_output).squeeze(1)

```

```

    avg_vector = sum_vector.float() / length_tensor.float()
    return avg_vector

    def forward(self, input_ids, attention_mask, token_type_ids, labels,
e1_mask, e2_mask):
        outputs = self.bert(input_ids, attention_mask=attention_mask,
                             token_type_ids=token_type_ids)
        sequence_output = outputs[0]
        pooled_output = outputs[1]
        # Average
        e1_h = self.entity_average(sequence_output, e1_mask)
        e2_h = self.entity_average(sequence_output, e2_mask)
        # Dropout -> tanh -> fc_layer
        pooled_output = self.cls_fc_layer(pooled_output)
        e1_h = self.e1_fc_layer(e1_h)
        e2_h = self.e2_fc_layer(e2_h)
        # Concat -> fc_layer
        concat_h = torch.cat([pooled_output, e1_h, e2_h], dim=-1)
        logits = self.label_classifier(concat_h)
        outputs = (logits,) + outputs[2:]
        # Softmax
        if labels is not None:
            if self.num_labels == 1:
                loss_fct = nn.MSELoss()
                loss = loss_fct(logits.view(-1), labels.view(-1))
            else:
                loss_fct = nn.CrossEntropyLoss()
                loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
            outputs = (loss,) + outputs
        return outputs

args.do_eval = True
if args.do_eval:
    trainer.load_model()
    res, test_loss, preds, labels = trainer.evaluate("test")

from sklearn.metrics import classification_report, confusion_matrix
predict = preds

```

```
true = labels
```

```
positive_labels = [i for i in range(len(rel2id)) if i != 0]
```

```
positive_names = [list(rel2id.keys())[list(rel2id.values()).index(i)] for i in  
range(len(rel2id)) if i != 0]
```

```
print(classification_report(  
    y_true=true,  
    y_pred=predict,  
    labels=positive_labels,  
    target_names=positive_names  
))
```

CNN

```
from __future__ import print_function
```

```
import numpy as np
```

```
np.random.seed(1337)
```

```
import gzip
```

```
import sys
```

```
if (sys.version_info > (3, 0)):
```

```
    import pickle as pkl
```

```
else:
```

```
    import cPickle as pkl
```

```
import keras
```

```
from keras.models import Model
```

```
from keras.models import load_model
```

```
from keras.layers import Input, Dense, Dropout, Activation, Flatten,  
concatenate
```

```
from keras.layers import Embedding
```

```
from keras.layers import Convolution1D, MaxPooling1D, GlobalMaxPooling1D
```

```
from keras.regularizers import Regularizer
```

```
from keras.preprocessing import sequence
```

```
batch_size = 64
```

```
nb_filter = 100
```

```
filter_length = 3
```

```

hidden_dims = 100
nb_epoch = 30
position_dims = 50

print("Load dataset")
f = gzip.open('/content/causal-relations.pkl.gz', 'rb')
data = pickle.load(f)
f.close()

embeddings = data['wordEmbeddings']
yTrain, sentenceTrain, positionTrain1, positionTrain2 = data['train_set']
yTest, sentenceTest, positionTest1, positionTest2 = data['test_set']

max_position = max(np.max(positionTrain1), np.max(positionTrain2))+1

n_out = max(yTrain)+1
max_sentence_len = sentenceTrain.shape[1]

print("sentenceTrain: ", sentenceTrain.shape)
print("positionTrain1: ", positionTrain1.shape)
print("yTrain: ", yTrain.shape)
print("sentenceTest: ", sentenceTest.shape)
print("positionTest1: ", positionTest1.shape)
print("yTest: ", yTest.shape)
print("Embeddings: ", embeddings.shape)

words_input = Input(shape=(max_sentence_len,), dtype='int32',
name='words_input')
words = Embedding(embeddings.shape[0], embeddings.shape[1],
weights=[embeddings], trainable=False)(words_input)

distance1_input = Input(shape=(max_sentence_len,), dtype='int32',
name='distance1_input')
distance1 = Embedding(max_position, position_dims)(distance1_input)

distance2_input = Input(shape=(max_sentence_len,), dtype='int32',
name='distance2_input')
distance2 = Embedding(max_position, position_dims)(distance2_input)

```

```

output = concatenate([words, distance1, distance2])

output = Convolution1D(filters=nb_filter,
                        kernel_size=filter_length,
                        padding='same',
                        activation='tanh',
                        strides=1)(output)

# we use standard max over time pooling
output = GlobalMaxPooling1D()(output)

output = Dropout(0.25)(output)
output = Dense(n_out, activation='softmax')(output)

model = Model(inputs=[words_input, distance1_input, distance2_input],
               outputs=[output])
model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam',
              metrics=['accuracy'])
model.summary()

print("Start training")

max_prec, max_rec, max_acc, max_f1 = 0,0,0,0

def getPrecision(pred_test, yTest, targetLabel):
    targetLabelCount = 0
    correctTargetLabelCount = 0

    for idx in range(len(pred_test)):
        if pred_test[idx] == targetLabel:
            targetLabelCount += 1

        if pred_test[idx] == yTest[idx]:
            correctTargetLabelCount += 1

    if correctTargetLabelCount == 0:
        return 0

```

```

    return float(correctTargetLabelCount) / targetLabelCount

def predict_classes(prediction):
    return prediction.argmax(axis=-1)

for epoch in range(nb_epoch):
    model.fit([sentenceTrain, positionTrain1, positionTrain2], yTrain,
              batch_size=batch_size, verbose=True, epochs=1)
    pred_test_ini = model.predict([sentenceTest, positionTest1, positionTest2],
                                  verbose=False)
    pred_test = predict_classes(pred_test_ini)

    dctLabels = np.sum(pred_test)
    totalDCTLabels = np.sum(yTest)

    acc = np.sum(pred_test == yTest) / float(len(yTest))
    max_acc = max(max_acc, acc)
    print("Accuracy: %.4f (max: %.4f)" % (acc, max_acc))

    f1Sum = 0
    f1Count = 0
    for targetLabel in range(1, max(yTest)):
        prec = getPrecision(pred_test, yTest, targetLabel)
        recall = getPrecision(yTest, pred_test, targetLabel)
        f1 = 0 if (prec+recall) == 0 else 2*prec*recall/(prec+recall)
        f1Sum += f1
        f1Count += 1

    macroF1 = f1Sum / float(f1Count)
    max_f1 = max(max_f1, macroF1)
    print("Non-other Macro-Averaged F1: %.4f (max: %.4f)\n" % (macroF1,
                                                                max_f1))

    pred_test_ini = model.predict([sentenceTest, positionTest1, positionTest2],
                                  verbose=False)
    pred_test = predict_classes(pred_test_ini)

```

```
model.save('causal_rel_model.h5')
```

UNSUPERVISED BERT

```
results = []
#timing stuff
start = time()
fit_time = 0
nearby_time = 0

batch_size = 50
num_cycles = 20
num_epochs = 3
num_class_epochs = 5
display_mod = 10
valid_mod = 50
neg_per = 15
neg_level = 2
num_nearby = 20
nearby_mod = 100
sample_power = .75
DH.scale_vocab_dist(sample_power)
total_step = 1
save_interval = 30 * 60
save_time = time()

for cycle in range(num_cycles):
    print("+"*80)
    print("+++ CYCLE %i +++" % cycle)
    print("+"*80)
    epoch = 0
    class_epoch = 0

    num_steps = DH.num_steps(batch_size)
```

```

for epoch in range(num_epochs):
    print("***80)
    print("=== UNSUP EPOCH %i ===" % epoch)
    print("***80)
    offset = 0 #if epoch else 400
    DH.shuffle_data()
        for step , batch in enumerate(DH.batches(batch_size, offset=offset,
neg_per=neg_per, neg_level=neg_level)):
            if not step: step = offset
            t0 = time()
            loss = drbertnn.partial_unsup_fit(*batch)
            fit_time = (fit_time * float(total_step) + time() - t0) / (total_step + 1) #
running average
            if step % display_mod == 0:
                m,s = divmod(time()-start, 60)
                h,m = divmod(m, 60)
                left = time_left(num_epochs, num_steps, fit_time, nearby_time, start,
nearby_mod)
                ml,sl = divmod(left, 60)
                hl,ml = divmod(ml, 60)
                pps = batch_size*(neg_per + 1) / fit_time
                print("(%i:%i:%i) step %i/%i, epoch %i Training Loss = %1.5f :: %0.3f
phrases/sec :: (%i:%i:%i) hours left"
                    % (h,m,s, step, num_steps, epoch, loss, pps, hl, ml, sl))
            if (total_step-1) % nearby_mod == 0:
                t0 = time()
                run_validation_test(num_nearby)
                valid_loss = bert.validation_loss(*DH.validation_batch())
print("Validation loss: %0.4f" % valid_loss)
                nearby_time = (nearby_time * float(total_step) + time() - t0) /
(total_step + 1) # running average
                total_step +=1
                valid_batch = DH.classification_batch(len(valid['labels']), valid['sdps'],
valid['targets'], valid['labels'])
                label_set = set(train['labels'])
                preds = bert.predict(valid_batch[0], valid_batch[1], valid_batch[3])
                cm, stats = eh.confusion_matrix(preds, valid['labels'], label_set)
                print("Saving model...")

```

```

model_file = bert.checkpoint()
result = {'cycle':cycle, 'unsup_epoch':epoch, 'class_epoch':class_epoch,
          'cm':cm, 'stats':stats, 'macro_f1':stats['macro_f1'], 'model':model_file}
results.append(result)
print("Macro P: %2.4f, R: %3.4f, F1: %0.4f" % (stats['macro_precision'],
stats['macro_recall'], stats['macro_f1']))

num_steps = len(train['labels']) // batch_size

for class_epoch in range(num_class_epochs):
    print("****80)
    print("**** CLASS EPOCH %i ****" % class_epoch)
    print("****80)
    for class_step in range(num_steps):
        inputs, targets, labels, lens = DH.classification_batch(batch_size,
train['sdps'], train['targets'], train['labels'], offset=class_step)
        class_batch = zip(inputs, targets, labels, lens)
        random.shuffle(class_batch)
        class_batch = zip(*class_batch)
        xent = bert.partial_class_fit(*class_batch)
        if class_step % display_mod == 0:
            m,s = divmod(time()-start, 60)
            h,m = divmod(m, 60)
            print("(%i:%i:%i) s %i/%i, e %i avg class xent loss = %0.4f" % (h,m,s,
class_step, num_steps, class_epoch, xent))
        if class_step % valid_mod == 0:
            valid_batch = DH.classification_batch(len(valid['labels']), valid['sdps'],
valid['targets'], valid['labels'])
            valid_xent = bert.validation_class_loss(*valid_batch)
m,s = divmod(time()-start, 60)
            h,m = divmod(m, 60)
            print("****80)
            print("(%i:%i:%i) s %i/%i, e %i validation avg class xent loss = %0.4f"
% (h,m,s, class_step, num_steps, class_epoch, valid_xent))
            print("****80)

label_set = set(train['labels'])
preds = bert.predict(valid_batch[0], valid_batch[1], valid_batch[3])

```

```

cm, stats = eh.confusion_matrix(preds, valid['labels'], label_set)
print("Saving model...")
model_file = bert.checkpoint()
result = {'cycle':cycle, 'unsup_epoch':epoch, 'class_epoch':class_epoch,
          'cm':cm, 'stats':stats, 'macro_f1':stats['macro_f1'], 'model':model_file}
results.append(result)
print("Macro P: %2.4f, R: %3.4f, F1: %0.4f" % (stats['macro_precision'],
stats['macro_recall'], stats['macro_f1']))
print("Done")
best = sorted(results, key=lambda x:x['macro_f1'], reverse=True)[0]
print("Best result %2.4f, C:%i, UE:%i, CE:%i" %
      (best['macro_f1'], best['cycle'], best['unsup_epoch'], best['class_epoch']))

```

```

with open('SemEval2010_task8_all_data/train_keys.txt', 'w') as f:
    i = 1
    for label in train['labels']:
        f.write("%s\t%s\n" % (i, int2label[label]))
    i += 1
with open('SemEval2010_task8_all_data/valid_keys.txt', 'w') as f:
    i = len(train['labels'])
    for label in valid['labels']:
        f.write("%s\t%s\n" % (i, int2label[label]))
    i += 1

```

```

valid_batch = DM.classification_batch(len(train['labels']), train['sdps'],
train['targets'], train['labels'])
preds = bert.predict(valid_batch[0], valid_batch[1], valid_batch[3],
return_probs=False)
with open('SemEval2010_task8_all_data/train_preds.txt', 'w') as f:
    i = 1
    for label in preds:
        f.write("%s\t%s\n" % (i, int2label[label]))
    i += 1

```

UNSUPERVISED RNN

```

num_epochs = 1
batch_size = 50
neg_per = 25
num_nearby = 50
nearby_mod = 50
sample_power = .75
DM.scale_vocab_dist(sample_power)
num_steps = DM.num_steps(batch_size)
total_step = 1
save_interval = 30 * 60 # half hour in seconds
save_time = time()

#timing stuff
start = time()
fit_time = 0
nearby_time = 0

for epoch in range(num_epochs):
    offset = 0
    DM.shuffle_data()
    for step, batch in enumerate(DM.batches(batch_size, offset=offset,
neg_per=neg_per)):
        if not step: step = offset
        t0 = time()
        loss = drnn.partial_unsup_fit(*batch)
    fit_time = (fit_time * float(total_step) + time() - t0) / (total_step + 1) # running
average
    if step % 10 == 0:
        m,s = divmod(time()-start, 60)
        h,m = divmod(m, 60)
        left = time_left(num_epochs, num_steps, fit_time, nearby_time, start,
nearby_mod)
        ml,sl = divmod(left, 60)
        hl,ml = divmod(ml, 60)
        pps = batch_size*(neg_per + 1) / fit_time
        print("(%i:%i:%i) step %i/%i, epoch %i Training Loss = %1.5f :: %0.3f
phrases/sec :: (%i:%i:%i) hours left"
            % (h,m,s, step, num_steps, epoch, loss, pps, hl, ml, sl))

```

```

    if (total_step-1) % nearby_mod == 0:
        t0 = time()
        run_validation_test(num_nearby)
        valid_loss = drnn.validation_loss(*DM.validation_batch())
        print("Validation loss: %0.4f" % valid_loss)
        nearby_time = (nearby_time * float(total_step) + time() - t0) / (total_step
+ 1) # running average

```

```

    if (time() - save_time) > save_interval:
        print("Saving model...")
        drnn.checkpoint()
        save_time = time()
    total_step += 1
drnn.checkpoint()

```

```

start = 200
stride = 100
end = start + stride
lowd = TSNE(perplexity=20, n_components=2, init='pca', n_iter=5000)
target_2d = lowd.fit_transform(word_embeddings[start:end])
fig, ax = plt.subplots(figsize=(28,16))
for i, label in enumerate(DM.vocab[start:end]):
    label = "%s" % (label)
    x, y = target_2d[i,:]
    ax.scatter(x, y, color='b')
    ax.annotate(label, xy=(x, y), xytext=(5, 2), textcoords='offset points',
                ha='right', va='bottom')
plt.savefig('image_DRNN.png', dpi=200)

```

```

zip_train = zip(train['raws'], train['sents'], train['sdps'], train['targets'],
train['labels'])
zip_valid = zip(valid['raws'], valid['sents'], valid['sdps'], valid['targets'],
valid['labels'])
zip_test = zip(test['raws'], test['sents'], test['sdps'], test['targets'])

```

```

for i, (raw, _, sdp, target, label) in enumerate(zip_train):
    if i > 5:
        break

```

```

print(raw)
    print("%s :: %s" % (DM.sequence_to_sentence(sdp, show_dep=True),
DM.sequence_to_sentence(target)))
print(int2label[label])
print("="*80)

```

COMPARISION

```

import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ['LSTM','RNN','CNN','RBERT','Causal_BERT','Unsup_BERT','Unsup_RNN']
y_pos = np.arange(len(objects))
performance = [68.6, 68, 97.75, 84.65, 84.14, 77 ,73.6]

plt.bar(y_pos, performance, align='center', alpha=1)
plt.xticks(y_pos, objects)
plt.ylabel('Usage')
plt.title('CE-Relation Accuracy plot')

plt.show()

```

B) Screenshot:

```
plt.plot(train_losses2)
plt.plot(validation_losses2)
plt.title("Cost of RE vs Epochs ")
plt.ylabel('Cost')
plt.xlabel('Epochs')
plt.legend(labels = ["Train","Validation"])
plt.show()
```

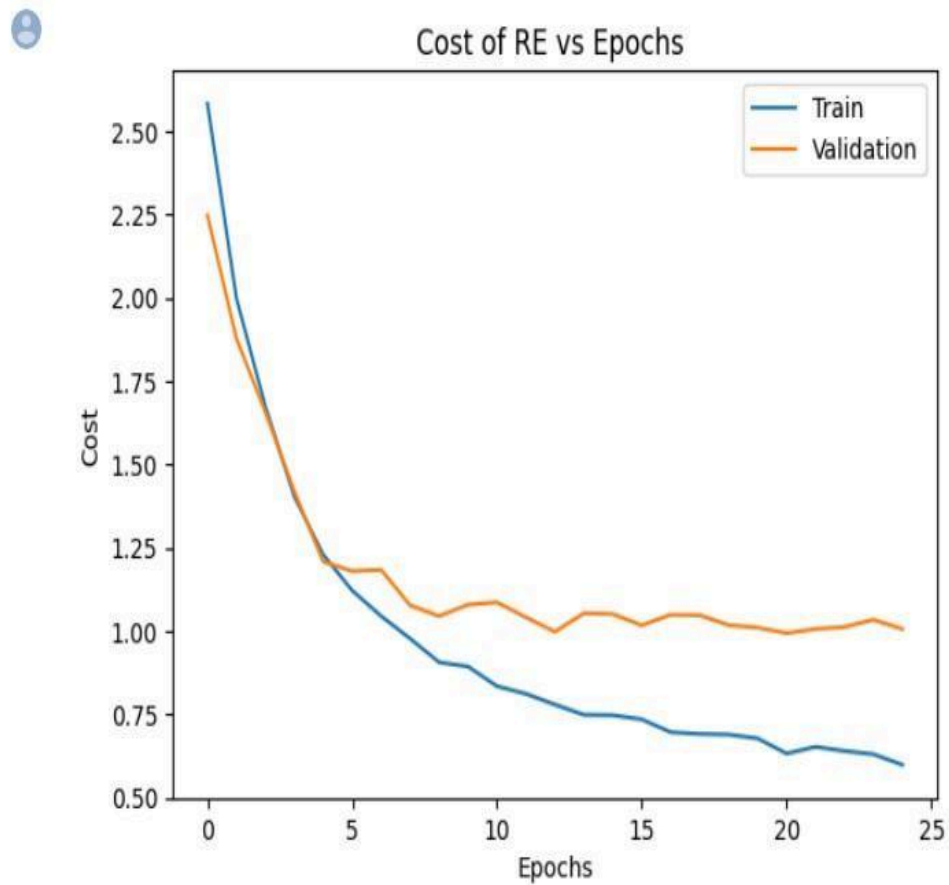


Fig.B.1: LSTM

```
plt.plot(train_acc_list)
plt.plot(test_acc_list)
plt.title("Accuracy of RE vs Epochs")
plt.ylabel('Acc(%)')
plt.xlabel('Epochs')
plt.legend(labels = ["Train", "Validation"])
plt.show()
```

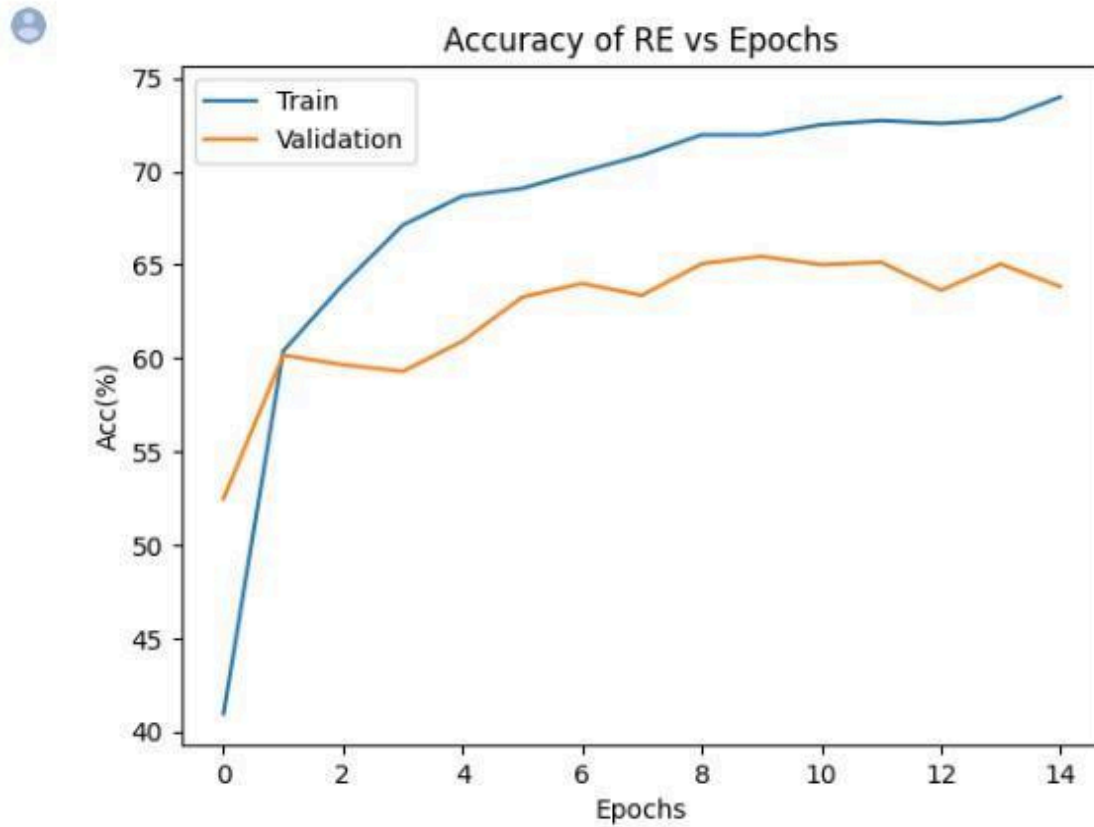


Fig.B.2: RNN

	Param #	Connected to
	0	[]
	0	[]
	0	[]
00)	6372900	['words_input[0][0]']
)	3200	['distance1_input[0][0]']
)	3200	['distance2_input[0][0]']
00)	0	['embedding[0][0]', 'embedding_1[0][0]', 'embedding_2[0][0]']
00)	120100	['concatenate[0][0]']
	0	['conv1d[0][0]']
	0	['global_max_pooling1d[0][0]']
	303	['dropout[0][0]']

Fig.B.3.CNN

```

1 from sklearn.metrics import classification_report, confusion_matrix
predict = preds
true = labels

positive_labels = [i for i in range(len(rel2id)) if i != 0]
positive_names = [list(rel2id.keys())[list(rel2id.values()).index(i)] for i in range(len(rel2id)) if i != 0]
print(classification_report(
    y_true=true,
    y_pred=predict,
    labels=positive_labels,
    target_names=positive_names
))

```

	precision	recall	f1-score	support
Cause-Effect(e1,e2)	0.00	0.94	0.92	134
Cause-Effect(e2,e1)	0.91	0.94	0.93	194
Component-Whole(e1,e2)	0.68	0.77	0.72	22
Component-Whole(e2,e1)	0.83	0.79	0.81	134
Content-Container(e1,e2)	0.85	0.89	0.87	108
Content-Container(e2,e1)	0.86	0.85	0.86	123
Entity-Destination(e1,e2)	0.87	0.92	0.89	153
Entity-Destination(e2,e1)	0.86	0.82	0.84	39
Entity-Origin(e1,e2)	0.89	0.88	0.89	211
Entity-Origin(e2,e1)	0.82	0.89	0.86	47
Instrument-Agency(e1,e2)	0.93	0.93	0.93	291
Instrument-Agency(e2,e1)	0.00	0.00	0.00	1
Member-Collection(e1,e2)	0.87	0.85	0.86	162
Member-Collection(e2,e1)	0.83	0.85	0.84	158
Message-Topic(e1,e2)	0.74	0.78	0.76	32
Message-Topic(e2,e1)	0.86	0.92	0.89	281
Other	0.87	0.93	0.90	218
Product-Producer(e1,e2)	0.86	0.94	0.90	51
Product-Producer(e2,e1)	0.00	0.00	0.00	0
micro avg	0.87	0.89	0.88	2263
macro avg	0.76	0.78	0.77	2263
weighted avg	0.87	0.89	0.88	2263

Fig.B.4.BERT

```

import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ['LSTM', 'RNN', 'CNN', 'Causal_BERT', 'Unsup_BERT', 'Unsup_RNN']
y_pos = np.arange(len(objects))
performance = [68.6, 68, 97.75, 84.14, 77, 73.6]

plt.bar(y_pos, performance, align='center', alpha=1)
plt.xticks(y_pos, objects)
plt.ylabel('Usage')
plt.title('CE-Relation Accuracy plot')

plt.show()

```

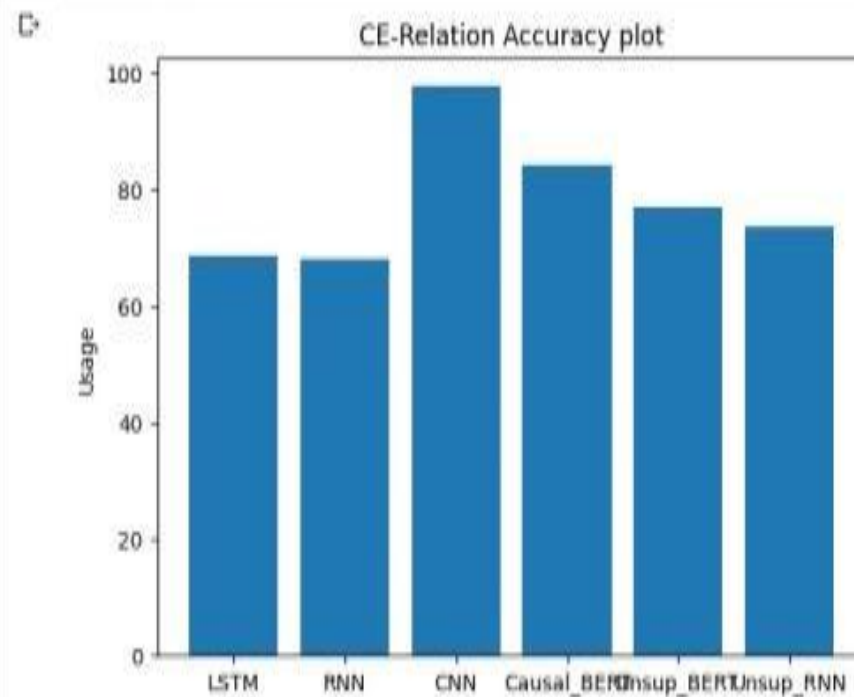


Fig.B.5.COMPARISON

C) RESEARCH PAPER

CAUSAL EVENT EXTRACTION

Akula Durga Sireesha

Sathyabama Institute of science
and Technology

Sirishaakula13@gmail.com

Addanki Teja kumar

Sathyabama Institue of Science
and Technology

Tejaaddanki29@gmail.com

Dr.M.Selvi B.E.,M.Tech.,Ph.D

Sathyabama Institute of Science
and Technology

Selvi.cse@sathyabama.ac.in

Abstract: The cause of the manner of causal extraction in herbal language is detection and Drawing motive and impact relationships among events or moves in the text. This purpose is to automatically become aware of useful cause-impact relationships Many applications which include know-how map creation and facts retrieval. Advanced deep studying algorithms together with LSTM and Causal BERT are also wanted. A style of techniques together with semantic coding. Become built a representation of motive and effect relationships between activities is the causal effect of the occasion approach of extraction. Because of the richness of natural language, extracting purpose and effect relationships is a tough task. A complex operation, but it can offer high-quality data the courting among occasions and activities. Supervised algorithms can extract cause-and-impact conclusions because they do not exist Training requires labeled records. But the set of rules does not paintings. They are performed and monitored on labeled records. We analyze the corpus created by using extending SemEval annotations. 2010 Question eight is given within the exam. We extract all reasons and consequences from the texts and keys of herbal languages Proper identification of "C" (purpose), "E" (effect) and tags is a critical a part of our work. "Emb" (Embedded Causation), means the semantic characteristic of causal events.

Keywords: Causal Event Extraction, LSTM, Language,

INTRODUCTION

The mission of figuring out purpose and impact relationships between occasions in herbal text the linguistic technique is referred to as causal occasion retrieval. Because it allows Identify enormous cause and effect relationships among

events to be used for choice making or further evaluation that is a very crucial assignment Information Extraction and Text Mining. Natural language processing methods Including element-of-speech tagging, dependency evaluation and component-covering Tagging is used to pick out gadgets and hyperlinks in textual content. Draw a reason and effect courting. Here are methods you could assist the matters within the text and the conjunctions that meet their purpose. We have exclusive tactics to locating reasons and extracting relationships and effects. Between texts of course. Using learning gadget learning models Annotation is a technique of identifying purpose and impact relationships in datasets. Linkage of products. Another approach is rule-based totally additionally, it seems at reason and effect relationships between users Set policies and examples. Finding and extracting cause and effect relationships of path use examples of strategies between templates in both eventualities Adaptation, organizational reputation, and dating extraction. There is information to be extracted, issues to be solved and choices to be made. Examples of applications for developing motive and effect relationships. Let's use that as an instance to decide the causes of scientific activities for analysis and treatment Diseases or reasons of forecasting marketplace financial tendencies.

OBJECTIVE

Determining motive and impact relationships of activities and expressing pairs of related occasions. And the purpose and impact relationship among them. This consists of growing automated techniques that are correct in extracting purpose and impact conclusions Identifying and extracting purpose and impact relationships from large volumes of text.

RELATED WORK

The version in Article [1] makes use of an independent bivariate design Transformed with LSTM-CRF (Conditional Random Field) embedding's Identify motive and impact relationships from the text. Bivariate sampling uses LSTM to version. Context and attention to choose up words and expressions. Guaranteed it is used to extract the impact of case coherence and CRF layer relationships. Upgrade The first-rate of phrase symbols additionally uses the interpretation of version concepts. The reason of the version is to separate causality into two According to the killed information.

Paper [2] dataset and version for computerized extraction of motive and impact relationships. Clinical capabilities are offered in the observe as MIMI Cause. MIMI-III 7,794 annotated databases include the dataset. Convolutional Neural Network networks (CNN) and lengthy-brief-time period reminiscence networks (LSTM). In the version, to express relationships between clinical standards and causation. To do It also makes use of a model to jointly expect the type and course of causality Multiple study techniques. The assessment results show that the proposed model outperforms present procedures to show motive and effect relationships from a medical textual content. Proposed transfer method and MIMI Cause dataset this will facilitate medical doctor decision-making and enhance the excellent of affected person care. Results

Article [3] studies proposes a deep model of an automatic neural network extracting purpose and effect relationships from linguistically knowledgeable text. Officer Accurate extraction of cause and impact relationships, syntactic composition and Semantic information in the form of dependency bushes and WorldNet hyperonyms. A model uses a bipartite LSTM network to capture contextual data a softmax classifier is used to expect the lifestyles and nature of purpose and impact relationships. Correlation The proposed model outperforms modern day models. Methods for extracting motive and effect relationships based on evaluation outcomes According to the killed statistics. This concept will have many packages; consisting of textual content mining, statistics extraction and expertise acquisition; It's clean to accept.

Article [4] suggests the proposed technique of automatically extracting the method Drug-associated activities (ADE) from scientific organic tissues. Officer ADE makes use of a unique extraction technique, external facts Resources which include clinical ontologies, dictionaries and semantic networks. It is called the identical object as the relation extraction module is used by the machine initially to know the medicines and associated sicknesses, their Correlation So this

device makes use of formal rule-primarily based understanding ADES may be primarily based on information assets and found drug-disorder associations. The set of rules then uses a device learning method to set ADs are divided into several companies in line with severity and chance. The effects of the assessment of the simple facts show the targets to offer higher than expertise based ADE extraction technique Latest era. Technology can help pharmacovigilance Work for better fitness and be affected person

Article [5] discusses tiers of Graph Convolutional Network (GCN) based paper A derivation of motive-effect relationship model is proposed. It uses the GCN version to seize Use syntactic and semantic relationships between phrases in a sentence these representations represent feasible motive and impact relationships. It is believed Causal relationships are refined the use of the second GCN level version. Run away with false positives. Develop accurate cause and effect relationships Extraction, get admission to understanding also includes outside goals Resources in the shape of scientific ontologies. The proposed version performs nicely Modern techniques of extracting motive and effect relationships, as proven via evaluation consequences According to the killed facts. Many packages including pharmacy tracking; Journal of Decision Making and Biomedical Research might also benefit precedent

Article [6] gives a detailed evaluation of the observe of advanced medical technology. Text occasion extraction technique. The trouble of extracting mass and so forth item recognition, stimulus occasion identification, and the argument about extraction is placed first within the article. He then discusses the take a look at various deep mastering frameworks which include Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and their variants are long Short Term Memory (LSTM) networks and Gate Repetition Units (GRU), which they are sure to return returned. The have a look at additionally examines how deep the studying is Models may be related to diverse outside facts sources Lexicons, ontologies and pre-trained language models must make matters better Extraction of results. The document concludes with a summary of several problems. Extraction packages such as organic research, herbal language Managing and studying social networks. Working researchers and clinicians Of route, the usage of deep mastering strategies to extract from textual content can carry splendid benefits From Bologna

Article [7] provides an in depth and in-intensity evaluation of inspection technologies. Methods for extracting motive and effect relationships from

biomedical textual content. Question Extraction of motive and impact relationships and lots of other responsibilities, incl Knowledge of purpose and impact of things and determination of course Cause and impact courting for the first time at paintings. Hence the thing makes diverse analyses deep getting to know frameworks along with Convolutional Neural Networks (CNN); Recurrent Neural Networks (RNN) and Graph Convolutional Networks (GCN); to explicit motive and effect relationships. The file also talks about integration Deep learning models for developing medical ontologies and dictionaries Extracting reason and impact relationships. The article concludes through offering there are many apps for extracting causes like Maecenas Pharmacovigilance, clinical selection making and biological studies. To Researchers and artists working on deep learning primarily based on reason. A survey is a useful tool for extracting relationships from biomedical textual content.

Article [8] introduces the SSA-UO look at, a totally new and unexpected approach. To understand the evaluation of Twitter data. It uses impartial mode Organization diagram (SOM) to organization words in line with frequency as one among a massive variety of tweets. Proximity Pair words as predetermined seed words with a demonstration of sense. Then use that sentence in base words. Officer With an extremely good sense of analysis, he additionally makes use of the ideal new word for design A semantic disambiguation technique based at the distribution of similarity terms. The consequences of assessment of baseline datasets display that SSA-UO Outperforms contemporary unsupervised sensitivity detection algorithms Twitter Analytics. The proposed approach is useful for plenty people Applications which include brand popularity control, patron remarks Analysis and tracking of social networks.

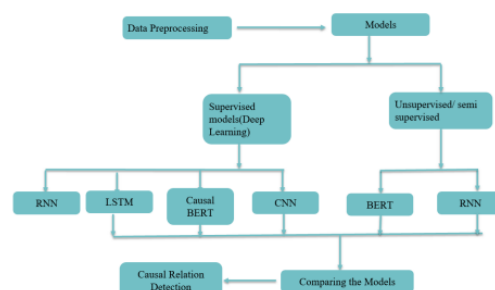
Article [9] placed college students into organizations with special dropout danger The take a look at suggests that classes are primarily based on the density of implicit mastering The art algorithm makes use of area to create a multi-dimensional function Student statistics such as demographics, performance and engagement The answers algorithm then makes use of a density-primarily based clustering approach to discover them Groups of students with positive behaviors that present a danger to each group Depending at the extent to which it is possibly to occur. Proposed the benefit of this method is that it does not require supervision can be used efficiently to kill named statistics and big records. Proposed this method is superior to different approaches currently used to assess dropout threat preaching according to results evaluated by means of student testimony. . The proposed guidelines will assist

educational institutions to discover danger groups. They assist college students improve their instructional fulfillment behavior

Article 10 inside the article is a way for robotically extracting causal relationships Links from biomedical texts are counseled. Using throwing strategies Traditional manager algorithms require categorized facts, which may be costly. It's time to attend to it in order that they spontaneously collect purpose and impact relationships; Teachers use texts. The steps within the technique are as follows: Drawing, drawing format and model order. Syntax and Semantics Analytical strategies are used to locate styles that propose cause and effect relationships. These sorts of conversation are then common and thru it is used to create and extract guests. That does the trick Better without the want to label the facts whilst checking out for importance Welcome to the textual content dataset.

SYSTEM ARCHITECTURE

The complete dataset was divided into checking out and education. Information the password is pre-processed from the dataset such as stop word elimination after the pre-processing, we started to create the fashions. Here is our plan 6 examples have been built. Four copies are given in Govt. And in 2 fashions uncontrolled records. We have advanced supervised fashions of RNN, LSTM, Causal BERT and CNN. We advanced BERT and RNN based on unsupervised models. Then we positioned the whole thing together we took the version very accurately and exactly and used this prototype He anticipated the event. In all of the above models, the variety is higher compared to the accuracy with all fashions. So, the use of CNN model, we predicted the result. Finally, if we provide Condemnation with organization predicts the relationship among the 2 all fashions with essence structure are without a doubt explained beneath.



SYSTEM REQUIREMENTS

Hardware Requirements:

- 8 GB RAM or higher
- 512 GB SSD ROM or higher

- Processor: Intel i5 8th gen, AMD Ryzen 5 or higher

Software Requirements

- Python
- Windows 10

MODULES AND THEIR DESCRIPTION

The test data and the train data is used to test and train the Models

	label	sentence
0	Component-Whole(e1,e1)	the system as described above has its greatest...
1	Other	the e1start child e1end was carefully wrapped ...
2	Instrument-Agency(e2,e1)	the e1start author e1end of a keygen uses a e2...
3	Other	a misty e1start ridge e1end uprises from the e...
4	Member-Collection(e1,e2)	the e1start student e1end e2start association ...
...	...	...
7995	Other	when the e1start notice e1end is sent by e2sta...
7996	Entity-Origin(e1,e2)	the e1start herbicide e1end is derived from a ...
7997	Entity-Destination(e1,e2)	to test this we placed a kitchen e1start match...
7998	Other	the farmers and city officials in the region h...
7999	Product-Producer(e2,e1)	the e1start surgeon e1end cuts a small e2start...

8000 rows x 2 columns

	label	sentence
0	Message-Topic(e1,e2)	the most common e1start audits e1end were abou...
1	Product-Producer(e2,e1)	the e1start company e1end fabricates plastic e...
2	Instrument-Agency(e2,e1)	the school e1start master e1end teaches the le...
3	Entity-Destination(e1,e2)	the suspect dumped the dead e1start body e1end...
4	Cause-Effect(e2,e1)	avian e1start influenza e1end is an infectious...
...	...	...
2712	Instrument-Agency(e2,e1)	after seating all the kids which itself takes...
2713	Product-Producer(e1,e2)	the minister attributed the slow production of...
2714	Component-Whole(e2,e1)	the e1start umbrella e1end e2start frame e2end...
2715	Product-Producer(e1,e2)	manos the hands of fate is a lowbudget horror ...
2716	Entity-Destination(e2,e1)	a few days before the service tom burris had t...

2717 rows x 2 columns

LSTM Model Implementation

The first implementation of the LSTM version. It does this by using taking enter and alerts to extract phrases Scoring is pre-figuring out the weightage of words. Intellectual facts from Wikipedia or many other resources of information kept inside the glove box It is described. Therefore, data is indicated and predicted. LSTM is similar to RNN. But in LSTM the weights are remembered however in RNN the gate is forgotten He did not take into account the burden of his earlier phrases.

```
class RelationClassification_Model2(nn.Module):
    def __init__(self, embedding_size, hidden_size, vocab_size, num_layers, class_num, pretrained_embedding, pad_idx):
        super(RelationClassification_Model2, self).__init__()

        self.embedding_size = embedding_size
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding.from_pretrained(pretrained_embedding, freeze = True, padding_idx = pad_idx)
        self.lstm = nn.LSTM(embedding_size, hidden_size, num_layers, bidirectional = True)
        self.linear = nn.Linear(2*hidden_size, class_num)
        self.softmax = nn.LogSoftmax(dim=1)
        self.dropout = nn.Dropout(0.7)

        def forward(self, x, mask_e1, mask_e2):
            embeddings = self.dropout(self.embedding(x))
            hidden, _ = self.lstm(embeddings)
            h = hidden.view(hidden.shape[0], hidden.shape[1], 2, self.hidden_size)
            e1, e2 = nn_avg_pooling(x, mask_e1, mask_e2)
            concats = torch.cat([e1, e2], dim = 1)
            outputs = self.softmax(self.linear(concats))

            return outputs

    def nn_avg_pooling(x, mask_e1, mask_e2):
        e1 = torch.zeros(x.shape[1], x.shape[0]).to(device)
        e2 = torch.zeros(x.shape[1], x.shape[0]).to(device)

        for i in range(x.shape[0]):
            sample = x[i, :, :].to(device)
            y = sample[mask_e1[i, :]].to(device)
            avg = torch.zeros(1, x.shape[1]).to(device)
            for j in range(y.shape[0]):
                avg += (torch.max(sample[j, :, :], y[j, :, :]) - avg).to(device) / (j + 1)
            e1[i, :] = avg

            y = sample[mask_e2[i, :]].to(device)
            avg = torch.zeros(1, x.shape[1]).to(device)
            for j in range(y.shape[0]):
                avg += (torch.max(sample[j, :, :], y[j, :, :]) - avg).to(device) / (j + 1)
            e2[i, :] = avg
```

```
torch.manual_seed(2005)

set_2_cat_class = dict(zip(categories(train_df['label']).cat.categories))

# Training and validation Data Generators
training_set = SemEval2010_TaskDataset2(train_df.partition(['train'], vocab))
pad_idx = 102400

# Parameters
params = {'batch_size': 256,
          'num_epochs': 100,
          'num_workers': 1,
          'validation_freq': 10}

training_generator = torch.utils.data.DataLoader(training_set, **params)
validation_set = SemEval2010_TaskDataset2(train_df.partition(['validation'], vocab))
validation_generator = torch.utils.data.DataLoader(validation_set, **params)

# Hyperparameters
embedding_size = 100
hidden_size = 100
vocab_size = len(vocab_df)
num_layers = 1
class_num = len(set_2_cat_class)
learning_rate = 1e-3
num_epochs = 10

# Model
model = RelationClassification_Model2(embedding_size, hidden_size, vocab_size, num_layers, class_num, embedding_pad_idx).to(device)

# Define loss function and Optimizer
criterion = nn.CrossEntropyLoss(ignore_index = pad_idx)
optimizer = optim.Adam(model.parameters())
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')

train_losses = []
validation_losses = []

train_accs = []
val_accs = []
```

RNN Model Implementation

It talks about enforcing an RNN version the usage of methods. Like a glove masking the first pre-processed text enter, replaces the words a vector of numbers. Then, a neural network (RNN) is dedicated to serial records processing, A vector of words belongs to the list. At every time step, the hidden layer of the RNN updates itself the country takes under consideration both the present day enter and the preceding hidden country. Consequently, the model can seize the context and relationships of the words in the sentence. After the RNN layer, greater complicated features can be extracted from the hidden country Add layers such as tight or dropped layers. Finally, we finish Predicts category probabilities in textual content using a dense layer with softmax implementation Office

```

torch.nn.LSTM(200)

int_cat_class = dict(enumerate(train_df['label'].cat.categories))

# Training and validation Data Generators
training_set = torch.utils.data.Dataset(train_df.partition(['train'], vocal))
val_set = torch.utils.data.Dataset(train_df.partition(['validation'], vocal))

pad_idx = 101704

# Parameters
params = {'batch_size': 256,
          'shuffle': True,
          'num_workers': 1,
          'collate_fn': BatchPadCollate(pad_idx=pad_idx)}

training_generator = torch.utils.data.DataLoader(training_set, **params)
validation_generator = torch.utils.data.DataLoader(validation_set, **params)

# Hyperparameters
embedding_size = 300
hidden_size = 150
vocab_size = len(vocab_df)
num_layers = 1
class_num = len(int_cat_class)
learning_rate = 1e-3
num_epochs = 15

# Model
model = RelationClassificationModel(embedding_size, hidden_size, vocab_size, num_layers, class_num, embedding_pad_idx).to(device)

# Define Loss Function and Optimizer
criterion = nn.L1Loss(ignore_index=pad_idx)
optimizer = optim.Adam(model.parameters())
lr_scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=10, min_lr=1e-5)

train_losses = []
validation_losses = []

train_accs = []
val_accs = []

```

```

def train():
    optimizer.step()

    # print statistics
    running_train_loss += loss.item()
    running_train_acc += acc.item()

    train_loss = torch.sum(torch.tensor(running_train_loss, dtype=torch.float)) / torch.sum(torch.tensor(train_labels, dtype=torch.float))

    if train_loss < 0.001:
        # print every 10 mini-batches
        print('Train loss after %i iterations in epoch %i: %f' % (N * train_loss + 1, epoch + 1, running_train_loss / 10))
        running_train_loss = 0.0

    train_acc = 0

    # Validation
    running_val_loss += loss.item()
    running_val_acc += acc.item()

    with torch.no_grad():
        for batch_data, batch_labels, batch_val_idx in torch.utils.data.iter_instances(validation_generator):
            # transfer to GPU
            batch_data, batch_labels, batch_val_idx = batch_data.to(device), batch_labels.to(device), batch_val_idx.to(device)

            # Model computation
            # forward + backward + optimize
            batch_outputs = model(batch_data, batch_val_idx)
            val_loss = criterion(batch_outputs, batch_labels)

            running_val_loss += val_loss.item()
            running_val_acc += acc.item()

            val_loss = torch.sum(torch.tensor(running_val_loss, dtype=torch.float)) / torch.sum(torch.tensor(batch_labels, dtype=torch.float))

            if val_loss < 0.001:
                # print every 10 mini-batches
                print('Val loss after %i iterations in epoch %i: %f' % (N * val_loss + 1, epoch + 1, running_val_loss / 10))
                running_val_loss = 0.0

            val_acc = 0

    train_loss.append(running_train_loss / train_loss)
    train_acc.append(train_acc / train_acc)
    validation_loss.append(running_val_loss / val_loss)
    val_acc.append(val_acc / val_acc)

```

Causal BERT Model Implementation

Describe the implementation of the BERT causal model, which first makes use of the BERT tokenizer to gain textual content information with sub words as a part of the pre-processing. After this, we run the Causal BERT layer on consecutive signatures. Each signal methods most effective the left context on the left facet of the right input sequence of the causal BERT layer. For this motive, an instance may additionally contain temporal relationships between the phrases of a sentence. After including extra layers which includes dense layer to extract greater complexes from the latent country, we use the remaining dense layer with softmax activation characteristic to output the expected class chances for the text.

```

eval_loss = eval_loss / nb_eval_steps
results = {'loss': eval_loss}
preds = np.argmax(preds, axis=-1)
write_predictions(
    self.args, os.path.join(self.args.eval_dir, "proposed_answers.txt"), preds
)

result = compute_metrics(preds, out_label_ids)
results.update(result)

print("Eval results")
for key in sorted(results.keys()):
    print(" {} = {:.4f}".format(key, results[key]))

return results, loss_eval, preds, out_label_ids

def save_model(self):
    if not os.path.exists(self.args.model_dir):
        os.makedirs(self.args.model_dir)
    model_to_save = {
        self.model.module if hasattr(self.model, "module") else self.model
    }
    model_to_save.save_pretrained(self.args.model_dir)
    torch.save(self.args, os.path.join(self.args.model_dir, "training_args.bin"))
    logger.info("Saving model checkpoint to %s", self.args.model_dir)
def load_model(self):
    if not os.path.exists(self.args.model_dir):
        raise Exception("Model doesn't exist! Train first!")

    try:
        self.args = torch.load(
            os.path.join(self.args.model_dir, "training_args.bin")
        )
        self.config = self.config_class.from_pretrained(self.args.model_dir)
        self.model = self.model_class.from_pretrained(
            self.args.model_dir, config=self.config, args=self.args
        )
        self.model.to(self.device)
        logger.info("Model loaded")
    except:
        raise Exception("Some files might be missing...")

```

```

eval_loss += tmp_eval_loss.mean().item()
loss_eval.append((tmp_eval_loss.mean().item()))
nb_eval_steps += 1

if preds is None:
    preds = logits.detach().cpu().numpy()
    out_label_ids = inputs["labels"].detach().cpu().numpy()
else:
    preds = np.append(preds, logits.detach().cpu().numpy(), axis=-1)
    out_label_ids = np.append(
        out_label_ids, inputs["labels"].detach().cpu().numpy(), axis=-1
    )

eval_loss = eval_loss / nb_eval_steps
results = {'loss': eval_loss}
preds = np.argmax(preds, axis=-1)
write_predictions(
    self.args, os.path.join(self.args.eval_dir, "proposed_answers.txt"), preds
)

result = compute_metrics(preds, out_label_ids)
results.update(result)

print("Eval results")
for key in sorted(results.keys()):
    print(" {} = {:.4f}".format(key, results[key]))

return results, loss_eval, preds, out_label_ids

def save_model(self):
    if not os.path.exists(self.args.model_dir):
        os.makedirs(self.args.model_dir)
    model_to_save = {
        self.model.module if hasattr(self.model, "module") else self.model
    }
    model_to_save.save_pretrained(self.args.model_dir)
    torch.save(self.args, os.path.join(self.args.model_dir, "training_args.bin"))
    logger.info("Saving model checkpoint to %s", self.args.model_dir)
def load_model(self):
    if not os.path.exists(self.args.model_dir):
        raise Exception("Model doesn't exist! Train first!")

```

CNN Model Implementation

He talks about the implementation of the CNN model. That word the order vector is first pre-processed after which a layer (CNN) is applied Extract nearby features from the enter array. Slipped via the filters given a listing of phrase vectors, the CNN calculates the layer of the detail among the filter out weights and the local phrase window at any time. As this step is repeated to extract a hard and fast of local gadgets from the enter array various filters. The softmax activation characteristic can be output thru the last dense layer Type of reality predicted within the textual content. Bundled and dense extra layers may be brought to extract greater complicated functions from the extract Local features.

```

4 from __future__ import print_function
import numpy as np
np.random.seed(1337)
import gzip

import sys
if (sys.version_info > (3, 0)):
    import pickle as pkl
else:
    import cPickle as pkl

import keras
from keras.models import Model
from keras.models import load_model
from keras.layers import Input, Dense, Dropout, Activation, Flatten, concatenate
from keras.layers import Embedding
from keras.layers import Convolution2D, MaxPooling2D, GlobalMaxPooling2D
from keras.regularizers import Regularizer
from keras.preprocessing import sequence

batch_size = 64
nb_filters = 100
filter_length = 3
hidden_dim = 100
nb_epoch = 30
position_dim = 50

print("Load dataset")
f = gzip.open('content/causal-relations.pkl.gz', 'rb')
data = pkl.load(f)
f.close()

embeddings = data['word_embeddings']
y_train, sentence_train, position_train2, position_train2 = data['train_set']
y_test, sentence_test, position_test1, position_test2 = data['test_set']

max_position = max(np.max(position_train1), np.max(position_train2)) + 1
n_out = max(y_train) + 1
max_sentence_len = sentence_train.shape[1]

print("sentence_train: ", sentence_train.shape)

```

```

print("position_train1: ", position_train1.shape)
print("position_train2: ", position_train2.shape)
print("y_train: ", y_train.shape)
print("sentence_test: ", sentence_test.shape)
print("position_test1: ", position_test1.shape)
print("y_test: ", y_test.shape)
print("embeddings: ", embeddings.shape)

words_input = Input(shape=(max_sentence_len,), dtype='int32', name='words_input')
words = Embedding(embeddings.shape[0], embeddings.shape[1], weights=[embeddings], trainable=False)(words_input)

distance_input = Input(shape=(max_sentence_len,), dtype='int32', name='distance_input')
distance = Embedding(max_position, position_dim)(distance_input)

distance_input = Input(shape=(max_sentence_len,), dtype='int32', name='distance_input')
distance1 = Embedding(max_position, position_dim)(distance_input)

output = concatenate([words, distance, distance1])

output = Convolution2D(filters=nb_filters,
                      kernel_size=filter_length,
                      padding='same',
                      activation='tanh',
                      strides=(1, 1))(output)

# we use standard max over time pooling
output = GlobalMaxPooling2D(output)

output = Dropout(0.25)(output)
output = Dense(n_out, activation='softmax')(output)

model = Model([words_input, distance_input, distance1_input], output=output)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

print("Start training")

max_grad, max_val, max_acc, max_f1 = 0, 0, 0, 0
def get_results(pred_test, y_test, target_label):
    target_label_count = 0
    correct_target_label_count = 0

```

```

result = {'cycle': cycle, 'unsup_epoch': epoch, 'class_epoch': class_epoch,
         'val_loss': val_loss, 'macro_f1': stats['macro_f1'], 'model': model_file}
results.append(result)
print("Macro F1: %2.4f, %2.4f, %2.4f, %2.4f" % (stats['macro_precision'], stats['macro_recall'], stats['macro_f1']))

num_steps = len(train['labels']) // batch_size

for class_epoch in range(num_class_epochs):
    print("====")
    print("==== CLASS EPOCH %d =====" % class_epoch)
    print("====")
    for class_step in range(num_steps):
        inputs, targets, labels, loss = DH.classification_batch(batch_size, train['words'], train['targets'], train['labels'], offset=class_step)
        class_batch = zip(inputs, targets, labels, loss)
        random.shuffle(class_batch)
        class_batch = zip(*class_batch)
        sent = bert.predict(class_batch)(class_batch)
        if class_step % display_step == 0:
            n_s = dimension(sent, 0)
            print("%d/%d" % (class_step, num_steps))
            if class_step % valid_step == 0:
                valid_batch = DH.classification_batch(len(valid['labels']), valid['words'], valid['targets'], valid['labels'])
                valid_sent = bert.predict(valid_batch)(valid_batch)
                n_v = dimension(valid_sent, 0)
                print("====")
                print("%d/%d" % (class_step, num_steps))
                print("====")
                print("Macro F1: %2.4f, %2.4f, %2.4f, %2.4f" % (stats['macro_precision'], stats['macro_recall'], stats['macro_f1']))
                best = sorted(results, key=lambda x: x['macro_f1'], reverse=True)[0]
                print("Best results %2.4f, %2.4f, %2.4f, %2.4f" % (best['macro_f1'], best['macro_precision'], best['macro_recall'], best['macro_f1']))

```

```

4 with open('SemEval2010_task8_all_data/train_keys.txt', 'w') as f:
    i = 1
    for label in train['labels']:
        f.write("%d\t%s\n" % (i, intlabel[label]))
        i += 1

with open('SemEval2010_task8_all_data/valid_keys.txt', 'w') as f:
    i = 1
    for label in valid['labels']:
        f.write("%d\t%s\n" % (i, intlabel[label]))
        i += 1

valid_batch = DH.classification_batch(len(train['labels']), train['words'], train['targets'], train['labels'])
preds = bert.predict(valid_batch[0], valid_batch[1], valid_batch[2], return_probs=False)
with open('SemEval2010_task8_all_data/train_preds.txt', 'w') as f:
    i = 1
    for label in preds:
        f.write("%d\t%s\n" % (i, intlabel[label]))
        i += 1

```

RNN Unsupervised Model Implementation

It talks approximately the advent of RNN. They suit the relationships between phrases inside the sentence are captured through the RNN layer. To encrypt we used automated text enter and truncation to go back to the authentic textual content Encoder losses permit education without supervision. Coded A passenger can be ordered into a predetermined quantity of clusters the usage of edges Methods After RNN-based totally auto encoder learns the ideal input sample to transcribe Text this permits finding corporations of texts which have semantic similarity. Finally, as opposed to popular cluster labels, we can assign cluster labels all documents. Where they cohere in articles with a similar distribution of phrases. And the outcome is predictable.

BERT Unsupervised Model Implementation

BERT talks about carvings. We have begun by analysing the text, we want to break it into sub-words using the BERT tokenizer. Unsupervised text category. Then he gave a practice model the following subscript is used for tokenized codes and creates an array Contextual embedding for each signal. Then, we follow embedding using the context sign. Methods which includes K, mode or hierarchical partitioning into indices how many bunches? It allows locating groups of words inside the input textual content. What is said in semantics? Finally, we train and expect the result the use of clustering functions as capabilities.

```

num_epochs = 1
batch_size = 10
neg_per = 25
num_nearby = 10
nearby_mod = 10
sample_power = 10
DH_scale_vocab_dist(sample_power)
num_steps = DH.num_steps(batch_size)
total_step = 1
save_interval = 10 * 60 # half hour in seconds
save_time = time()

storing stuff
start = time()
fit_time = 0
nearby_time = 0

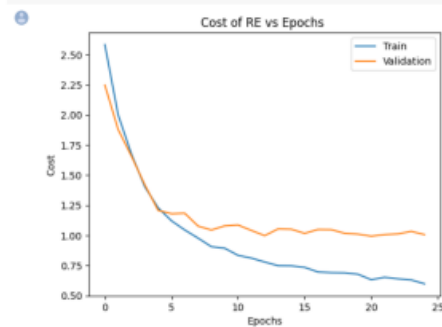
for epoch in range(num_epochs):
    offset = 0
    DH.shuffle_data()
    for step, batch in enumerate(DH.batches(batch_size, offset=offset, neg_per=neg_per)):
        if not step: step = offset
        offset += batch_size
        t0 = time()
        loss = dnn.partial_unsup_fit(batch)
        fit_time = (fit_time + time() - t0) / (total_step + 1) # running average
        if step % 10 == 0:
            t1 = time()
            h1 = dnn.get(w, 0)
            h2 = dnn.get(w, 0)
            left = time_left(num_epochs, num_steps, fit_time, nearby_time, start, nearby_mod)
            h1, l1 = dnn.get(w, 0)
            h2, l2 = dnn.get(w, 0)
            pps = batch_size * (neg_per + 1) / fit_time
            print("({N}/{N}) step {N}/{N}, epoch {N} training loss = {N:.5f} :: {N:.3f} phrases/sec :: ({N}/{N}) hours left"
                  " {N}/{N}, step, num_steps, epoch, loss, pps, h1, l1, l2)
            if (total_step - 1) % nearby_mod == 0:
                t0 = time()
                run_validation_test(num_nearby)
                valid_loss = dnn.validation_loss(DH.validation_batch())
                print("Validation loss: {N:.4f} {N} valid_loss)
                nearby_time = (nearby_time + time() - t0) / (total_step + 1) # running average
            if (time() - save_time) > save_interval:
                print("Saving model...")
                dnn.checkpoint()

```

```

plt.plot(train_losses2)
plt.plot(validation_losses2)
plt.title("Cost of RE vs Epochs ")
plt.ylabel("Cost")
plt.xlabel("Epochs")
plt.legend(labels = ["Train", "Validation"])
plt.show()

```



```

start = 200
stride = 100
end = start + stride
load = TSNL(perplexity=20, n_components=2, init='pca', n_iter=5000)
target_2d = load.fit_transform(word_embeddings[start:end])
fig, ax = plt.subplots(figsize=(10,10))

for i, label in enumerate(DH.vocab[start:end]):
    label = "%s" % label
    x, y = target_2d[i,:]
    ax.scatter(x, y, color="b")
    ax.annotate(label, xy=(x, y), xytext=(5, 2), textcoords='offset points',
               ha='right', va='bottom')
plt.savefig("image_2000.png", dpi=300)

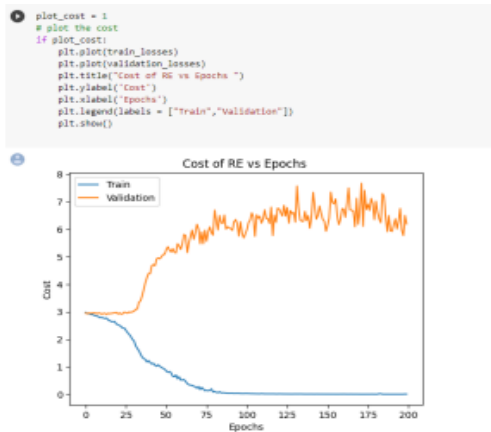
[ ] sig_train = sig(train['raw'], train['sents'], train['dps'], train['targets'], train['labels'])
sig_valid = sig(valid['raw'], valid['sents'], valid['dps'], valid['targets'], valid['labels'])
sig_test = sig(test['raw'], test['sents'], test['dps'], test['targets'])

[ ] for i, (raw, sdp, target, label) in enumerate(sig_train):
    if i > 5:
        break
    print(raw)
    print("%s :: %s" % (DH.sequence_to_sentence(sdp, show_deps=True), DH.sequence_to_sentence(target)))
    print(int2label[label])
    print("\n")

```

RESULT AND DISCUSSION

The graph underneath shows that all facts is divided into elements: schooling and trying out. Dates for both dates are proven right here.



SUPERVISED MODELS

LSTM Model

The graph beneath shows age and statistics loss for fitness and marketing Check up to age 25 There is greater statistics loss at the start up age As age slowly will increase, facts loss decreases. Uses many Over time, the version will become more knowledgeable and greater relevant.

In this example the diploma of accuracy is calculated as zero.6430807820742049. It suggests that about sixty four.31% of the predicted causal hyperlinks are accurate. Obsessive Mera Dilai Edoi Osenka Ravana zero.6894858053578062 68.95% of genuine causal links have been effectively detected. Accuracy indicator for this the score is zero.6915715863084284, which represents approximately 69.60% of all instances. The data set contained correct predictions made through the version. F1 score for this rating 0.6574352187234178, which suggests the general overall performance of the version adjusting for both accuracy and remarks.

```
[ ] precision_metric3, recall_metric3, accuracy_metric3, f1_metric3
```

```

(0.6430807820742049,
 0.6894858053578062,
 0.6915715863084284,
 0.6574352187234178)

```

Accuracy assesses the accuracy of predictions, taking the whole thing into consideration Categories. The accuracy become determined to be zero.68, i.e. 68% of the cases given the facts predicted the model accurately. Macro Average: Macro average offers same weight to each category within the calculation. Average precision, do not forget and F1 score throughout all training. Careful, remember and F1-. They have macro scores of 0.66, zero. Sixty three and 0.Sixty four respectively. Section: Recall and F1 score for all when calculating mean statement Classes get the common imply (variety of occurrences) for every Various precision, bear in mind and F1 score averaged zero.68, 0.68 and 0.Sixty eight respectively.

```

import seaborn as sns
print(classification_report(y_test2, y_predictions2))

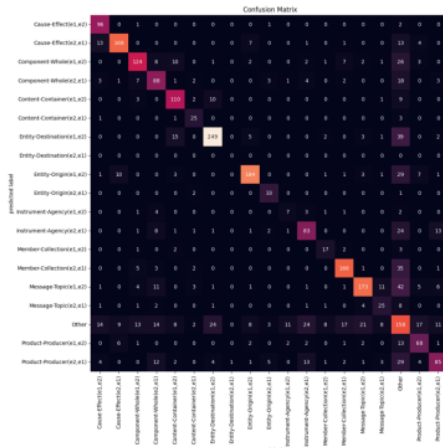
labels = list(int_2_cat_class.values())
f, ax = plt.subplots(figsize=(10, 12))
cmf_matrix = confusion_matrix(y_test2, y_predictions2)
ax = sns.heatmap(cmf_matrix.T, square=True, annot=True, fmt='d', char=False, xticklabels=labels, yticklabels=labels)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title('Confusion Matrix')
plt.show()

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score were zero in some classes.
  _warn_gf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score were zero in some classes.
  _warn_gf(average, modifier, msg_start, len(result))
precision    recall    f1-score   support

0         0.96    0.72    0.82    134
1         0.81    0.87    0.84    104
2         0.66    0.77    0.71    162
3         0.66    0.59    0.62    150
4         0.81    0.72    0.76    155
5         0.83    0.64    0.72    39
6         0.79    0.86    0.82    291
7         0.89    0.80    0.84    1
8         0.76    0.87    0.81    211
9         0.92    0.79    0.86    47
10        0.37    0.32    0.34    22
11        0.41    0.62    0.51    134
12        0.58    0.53    0.56    32
13        0.79    0.83    0.81    261
14        0.87    0.82    0.84    210
15        0.57    0.49    0.53    91
16        0.43    0.75    0.58    404
17        0.49    0.43    0.46    286
18        0.51    0.49    0.50    123

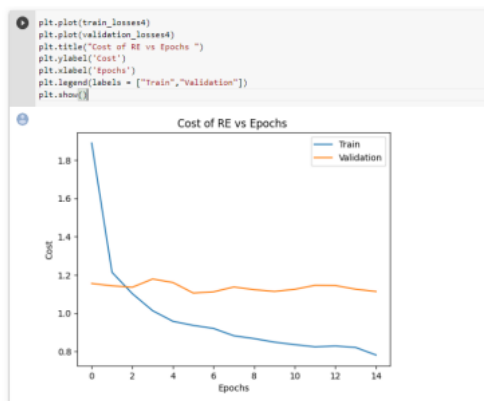
accuracy    0.68    0.68    0.68    2717
macro avg   0.66    0.63    0.64    2717
weighted avg 0.68    0.68    0.68    2717

```

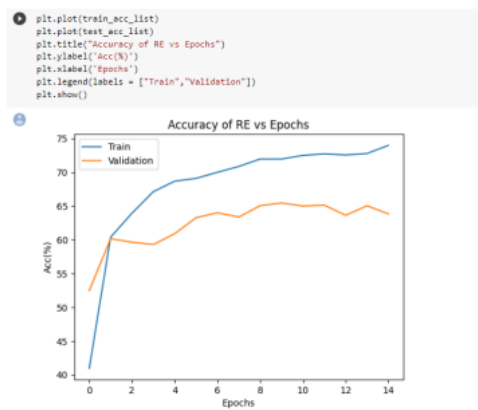


RNN Model

Data loss for education and trying out epochs up to 14 epochs is proven in Fig. The diagram under. Data loss is excessive inside the early epoch; however, whilst as the epoch step by step will increase, records loss decreases. These are centuries of assist the version is surprisingly practical and accurate.



The graph below suggests the truthful period and their accuracy set up check and validation records. Accuracy also will increase with age. Escalation is a totally found out example. 14 We came to be destroyed.



Precision: In this situation, the value of Precision_metric3 is zero.5975, which means approx. 59.79% of expected advantageous cases virtually occur. With a value of Recall_metric3 0.6177, the RNN model correctly detected 61. Seventy seven% real positives. Captain. An accuracy_metric3 value of 0.6445 shows common accuracy the performance of RNN version is set 64. Forty five%. F1 score is balanced score Model overall performance, which is calculated as a harmonious common of keep in mind and precision. It is calculated as $2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$. F1_metric3 value a cost of 0.6009 shows that the version plays equally nicely in terms of don't forget and precision.

```
[ ] precision_metric3, recall_metric3, accuracy_metric3, f1_metric3
```

```

(0.5975402899575979,
 0.6176925594152751,
 0.644608023555392,
 0.600836951894597)

```

```

import seaborn as sns
print(classification_report(y_test2, y_predictions2))

labels = list(int_2_cat_class.values())
f, ax = plt.subplots(figsize=(10, 12))
cmf_matrix = confusion_matrix(y_test2, y_predictions2)
ax = sns.heatmap(cmf_matrix.T, square=True, annot=True, fmt='d', char=False, xticklabels=labels, yticklabels=labels)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title('Confusion Matrix')
plt.show()

```

Accuracy reflects the general capacity of the version to make correct predictions in any respect degrees. It measures the share of efficaciously anticipated cases Samples. In this case the RNN model achieves 68% accuracy on the dataset. Zero.68 typical accuracy. Measure the weights of both lessons similarly while calculating the common overall performance throughout all training. He makes an example in trendy, for all training, it is exactly clean from the common macros, I don't forget; and F1 rating, every around zero.66.

Taking under consideration the squared distribution of the statistics set, the weighted suggest cost is determined Average overall performance indicators for all instructions. In those instances, for the

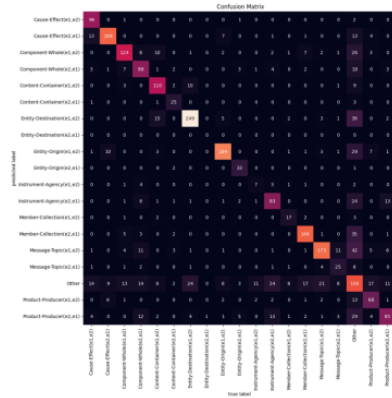
duration of instructions Unbalanced, he offers greater representational dimensions. A weighted average value the accuracy, completeness and F1 index are near 0. Sixty eight, which shows the index of RNN version. General productivity, taking into consideration the distribution of instructions.

```

$ /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:13
_warn_prf(average, modifier, msg_start, len(result))
$ /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:13
_warn_prf(average, modifier, msg_start, len(result))
$ /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:13
_warn_prf(average, modifier, msg_start, len(result))
precision recall f1-score support
0 0.96 0.72 0.82 134
1 0.81 0.87 0.84 194
2 0.66 0.77 0.71 162
3 0.66 0.59 0.62 150
4 0.81 0.72 0.76 153
5 0.83 0.64 0.72 39
6 0.79 0.86 0.82 291
7 0.80 0.80 0.80 1
8 0.76 0.87 0.81 211
9 0.52 0.70 0.60 47
10 0.37 0.32 0.34 22
11 0.81 0.62 0.71 134
12 0.68 0.53 0.60 32
13 0.78 0.83 0.80 201
14 0.67 0.82 0.74 210
15 0.57 0.49 0.53 51
16 0.43 0.35 0.38 454
17 0.69 0.63 0.66 108
18 0.51 0.69 0.59 123

accuracy 0.68 2717
macro avg 0.66 0.64 2717
weighted avg 0.68 0.68 2717

```



BERT causal version

The accuracy of the version based at the experimental facts is eighty four.14%, as shown by the precision zero.8414. Recall the accuracy and efficiency of the opposition as proven within the F1 score. 0.8414. The fee of the loss characteristic is determined in the evaluation manner Indicates loss. It calculates the distinction between expected effects Signs additionally appear. In this example the loss is 1.0142. These assessment results provide a comprehensive evaluation model. Performance on the test information set. The model seems to predict nicely High precision and effectively label the project causal fabric extraction according to F1 The loss price shows how properly the test version can reduce the difference Actual labels and expected results in training

```

[ ] args.do_eval = True
if args.do_eval:
    trainer.load_model()
    res, test_loss, preds, labels = trainer.evaluate("test")

Evaluating: 0% | 0/85 [00:00<, ?it/s]***** Running evaluation on %s dataset ***** test
Num examples = %d 2717
Batch size = %d 32
Evaluating: 100% 85/85 [00:30:00:00, 2.74it/s]***** Eval results *****
acc = 0.8414
f1 = 0.8414
loss = 1.0142

```

CNN Model

In the enter layer report input, distance1_input and distance2_input. They are sentences, first degree and second degree respectively. Embed_1 for the first stage and embed_2 for the second level. Place is the achievement of three orders. An integrated layer is used to connect those\ Linked embedding's are subjected to a one-dimensional layer transformation. (Change 1d). Dimensionality is decreased the use of a 1D Global Max layer. (global_max_pooling1d). A dropout layer is used to keep away from redundancy. The output runs thru the dense layer to represent the three classes. With three gadgets. A general of 126,803 parameters are trainable. 499 703 in VI version.

```

) Load dataset
sentenceTrain: (8000, 97)
positionTrain: (8000, 97)
yTrain: (8000,)
sentenceTest: (2717, 97)
positionTest: (2717, 97)
yTest: (2717,)
Embedding: (21243, 300)
Model: "model"

Layer (type) Output Shape Param # Connected to
-----
words_input (InputLayer) [(None, 97)] 0 []
distance1_input (InputLayer) [(None, 97)] 0 []
distance2_input (InputLayer) [(None, 97)] 0 []
embedding (Embedding) (None, 97, 300) 8572900 ['words_input[0][0]']
embedding_1 (Embedding) (None, 97, 50) 3200 ['distance1_input[0][0]']
embedding_2 (Embedding) (None, 97, 50) 3200 ['distance2_input[0][0]']
concatenate (Concatenate) (None, 97, 400) 0 ['embedding[0][0]', 'embedding_1[0][0]', 'embedding_2[0][0]']
conv1d (Conv1D) (None, 97, 100) 120100 ['concatenate[0][0]']
global_max_pooling1d (GlobalMaxPooling1D) (None, 100) 0 ['conv1d[0][0]']
dropout (Dropout) (None, 100) 0 ['global_max_pooling1d[0][0]']
dense (Dense) (None, 3) 303 ['dropout[0][0]']

Total params: 8,499,703
Trainable params: 126,803

```

The model has 126,803 out of 6,499,703 trainable parameters. That is, the version is tested the use of check data after schooling using schooling facts. Accuracy and other macro-averaged F1 rankings are typed after each exercise. In addition to the epoch, the outcomes of most accuracy and F1 ratings are proven. Version Shown by using High, it appears to do a very good process of drawing cause and effect relationships Accuracy and F1 ratings. However, let them take into account that according to the movement and context some aspects of architectural fashions, training techniques and assessment metrics can trade.

```

Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 17s 136ms/step - loss: 0.0012 - accuracy: 1.0000
Accuracy: 0.9093 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 17s 136ms/step - loss: 0.0011 - accuracy: 1.0000
Accuracy: 0.9095 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 18s 146ms/step - loss: 0.0010 - accuracy: 1.0000
Accuracy: 0.9089 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 17s 135ms/step - loss: 7.2164e-04 - accuracy: 1.0000
Accuracy: 0.9095 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 18s 146ms/step - loss: 7.6582e-04 - accuracy: 1.0000
Accuracy: 0.9095 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 18s 146ms/step - loss: 0.0131e-04 - accuracy: 1.0000
Accuracy: 0.9089 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9255 (max: 0.9302)

125/125 [=====] - 17s 135ms/step - loss: 5.4280e-04 - accuracy: 1.0000
Accuracy: 0.9095 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9213 (max: 0.9302)

125/125 [=====] - 18s 146ms/step - loss: 4.9125e-04 - accuracy: 1.0000
Accuracy: 0.9089 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9255 (max: 0.9302)

125/125 [=====] - 17s 135ms/step - loss: 5.1845e-04 - accuracy: 1.0000
Accuracy: 0.9095 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9213 (max: 0.9302)

125/125 [=====] - 17s 135ms/step - loss: 3.7852e-04 - accuracy: 1.0000
Accuracy: 0.9178 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

125/125 [=====] - 17s 135ms/step - loss: 3.4472e-04 - accuracy: 1.0000
Accuracy: 0.9095 (max: 0.9016)
Non-other Macro-Averaged F1: 0.9170 (max: 0.9302)

```

UNSUPERVISED MODELS

Unsupervised Bert

In this version, we calculated the remember accuracy and familiarity of all fl statements. In the information set. The accuracy of the cause-effect courting is 93.Sixty eight%, i.e. of all reason-effect instances, 93.68% were efficiently expected. Component-integer courting, do not forget seventy three.03%, 73.03% i.e. The cases that Component-Toti should have had are accurate anticipated. The F1 rating for the content material dating is 86.73%, which shows there is often a robust balance among don't forget and precision. Except for the "_Other" elegance, the implementation of the complete model is taken into consideration. Taking under consideration the calculation of the small mean effect. Calculates precision, take into account, and F1 primarily based on the variety of fake positives, false positives, and false positives. Denies all relationships. The average F1 is small at 80.Sixty nine%. Precision, keep in mind, and F1 rankings had been averaged for every setting, except the "_Other" elegance is the result of the chat within the macro. No matter what Relationships arise greater regularly, which gives them equal weight. Average F1 motive force this case is 80.47%. Official evaluation based totally on record F1 macro imply rating is 80.Forty seven% and specific (9+1) path Directional estimation is taken into consideration.

```

Confusion matrix:
      C=0  C=1  C=2  C=3  C=4  C=5  C=6  C=7  C=8  C=9  C=10  C=11  C=12  C=13  C=14  C=15  C=16  C=17  C=18  C=19  C=20  C=21  C=22  C=23  C=24  C=25  C=26  C=27  C=28  C=29  C=30  C=31  C=32  C=33  C=34  C=35  C=36  C=37  C=38  C=39  C=40  C=41  C=42  C=43  C=44  C=45  C=46  C=47  C=48  C=49  C=50  C=51  C=52  C=53  C=54  C=55  C=56  C=57  C=58  C=59  C=60  C=61  C=62  C=63  C=64  C=65  C=66  C=67  C=68  C=69  C=70  C=71  C=72  C=73  C=74  C=75  C=76  C=77  C=78  C=79  C=80  C=81  C=82  C=83  C=84  C=85  C=86  C=87  C=88  C=89  C=90  C=91  C=92  C=93  C=94  C=95  C=96  C=97  C=98  C=99  C=100  C=101  C=102  C=103  C=104  C=105  C=106  C=107  C=108  C=109  C=110  C=111  C=112  C=113  C=114  C=115  C=116  C=117  C=118  C=119  C=120  C=121  C=122  C=123  C=124  C=125  C=126  C=127  C=128  C=129  C=130  C=131  C=132  C=133  C=134  C=135  C=136  C=137  C=138  C=139  C=140  C=141  C=142  C=143  C=144  C=145  C=146  C=147  C=148  C=149  C=150  C=151  C=152  C=153  C=154  C=155  C=156  C=157  C=158  C=159  C=160  C=161  C=162  C=163  C=164  C=165  C=166  C=167  C=168  C=169  C=170  C=171  C=172  C=173  C=174  C=175  C=176  C=177  C=178  C=179  C=180  C=181  C=182  C=183  C=184  C=185  C=186  C=187  C=188  C=189  C=190  C=191  C=192  C=193  C=194  C=195  C=196  C=197  C=198  C=199  C=200  C=201  C=202  C=203  C=204  C=205  C=206  C=207  C=208  C=209  C=210  C=211  C=212  C=213  C=214  C=215  C=216  C=217  C=218  C=219  C=220  C=221  C=222  C=223  C=224  C=225  C=226  C=227  C=228  C=229  C=230  C=231  C=232  C=233  C=234  C=235  C=236  C=237  C=238  C=239  C=240  C=241  C=242  C=243  C=244  C=245  C=246  C=247  C=248  C=249  C=250  C=251  C=252  C=253  C=254  C=255  C=256  C=257  C=258  C=259  C=260  C=261  C=262  C=263  C=264  C=265  C=266  C=267  C=268  C=269  C=270  C=271  C=272  C=273  C=274  C=275  C=276  C=277  C=278  C=279  C=280  C=281  C=282  C=283  C=284  C=285  C=286  C=287  C=288  C=289  C=290  C=291  C=292  C=293  C=294  C=295  C=296  C=297  C=298  C=299  C=300  C=301  C=302  C=303  C=304  C=305  C=306  C=307  C=308  C=309  C=310  C=311  C=312  C=313  C=314  C=315  C=316  C=317  C=318  C=319  C=320  C=321  C=322  C=323  C=324  C=325  C=326  C=327  C=328  C=329  C=330  C=331  C=332  C=333  C=334  C=335  C=336  C=337  C=338  C=339  C=340  C=341  C=342  C=343  C=344  C=345  C=346  C=347  C=348  C=349  C=350  C=351  C=352  C=353  C=354  C=355  C=356  C=357  C=358  C=359  C=360  C=361  C=362  C=363  C=364  C=365  C=366  C=367  C=368  C=369  C=370  C=371  C=372  C=373  C=374  C=375  C=376  C=377  C=378  C=379  C=380  C=381  C=382  C=383  C=384  C=385  C=386  C=387  C=388  C=389  C=390  C=391  C=392  C=393  C=394  C=395  C=396  C=397  C=398  C=399  C=400  C=401  C=402  C=403  C=404  C=405  C=406  C=407  C=408  C=409  C=410  C=411  C=412  C=413  C=414  C=415  C=416  C=417  C=418  C=419  C=420  C=421  C=422  C=423  C=424  C=425  C=426  C=427  C=428  C=429  C=430  C=431  C=432  C=433  C=434  C=435  C=436  C=437  C=438  C=439  C=440  C=441  C=442  C=443  C=444  C=445  C=446  C=447  C=448  C=449  C=450  C=451  C=452  C=453  C=454  C=455  C=456  C=457  C=458  C=459  C=460  C=461  C=462  C=463  C=464  C=465  C=466  C=467  C=468  C=469  C=470  C=471  C=472  C=473  C=474  C=475  C=476  C=477  C=478  C=479  C=480  C=481  C=482  C=483  C=484  C=485  C=486  C=487  C=488  C=489  C=490  C=491  C=492  C=493  C=494  C=495  C=496  C=497  C=498  C=499  C=500  C=501  C=502  C=503  C=504  C=505  C=506  C=507  C=508  C=509  C=510  C=511  C=512  C=513  C=514  C=515  C=516  C=517  C=518  C=519  C=520  C=521  C=522  C=523  C=524  C=525  C=526  C=527  C=528  C=529  C=530  C=531  C=532  C=533  C=534  C=535  C=536  C=537  C=538  C=539  C=540  C=541  C=542  C=543  C=544  C=545  C=546  C=547  C=548  C=549  C=550  C=551  C=552  C=553  C=554  C=555  C=556  C=557  C=558  C=559  C=560  C=561  C=562  C=563  C=564  C=565  C=566  C=567  C=568  C=569  C=570  C=571  C=572  C=573  C=574  C=575  C=576  C=577  C=578  C=579  C=580  C=581  C=582  C=583  C=584  C=585  C=586  C=587  C=588  C=589  C=590  C=591  C=592  C=593  C=594  C=595  C=596  C=597  C=598  C=599  C=600  C=601  C=602  C=603  C=604  C=605  C=606  C=607  C=608  C=609  C=610  C=611  C=612  C=613  C=614  C=615  C=616  C=617  C=618  C=619  C=620  C=621  C=622  C=623  C=624  C=625  C=626  C=627  C=628  C=629  C=630  C=631  C=632  C=633  C=634  C=635  C=636  C=637  C=638  C=639  C=640  C=641  C=642  C=643  C=644  C=645  C=646  C=647  C=648  C=649  C=650  C=651  C=652  C=653  C=654  C=655  C=656  C=657  C=658  C=659  C=660  C=661  C=662  C=663  C=664  C=665  C=666  C=667  C=668  C=669  C=670  C=671  C=672  C=673  C=674  C=675  C=676  C=677  C=678  C=679  C=680  C=681  C=682  C=683  C=684  C=685  C=686  C=687  C=688  C=689  C=690  C=691  C=692  C=693  C=694  C=695  C=696  C=697  C=698  C=699  C=700  C=701  C=702  C=703  C=704  C=705  C=706  C=707  C=708  C=709  C=710  C=711  C=712  C=713  C=714  C=715  C=716  C=717  C=718  C=719  C=720  C=721  C=722  C=723  C=724  C=725  C=726  C=727  C=728  C=729  C=730  C=731  C=732  C=733  C=734  C=735  C=736  C=737  C=738  C=739  C=740  C=741  C=742  C=743  C=744  C=745  C=746  C=747  C=748  C=749  C=750  C=751  C=752  C=753  C=754  C=755  C=756  C=757  C=758  C=759  C=760  C=761  C=762  C=763  C=764  C=765  C=766  C=767  C=768  C=769  C=770  C=771  C=772  C=773  C=774  C=775  C=776  C=777  C=778  C=779  C=780  C=781  C=782  C=783  C=784  C=785  C=786  C=787  C=788  C=789  C=790  C=791  C=792  C=793  C=794  C=795  C=796  C=797  C=798  C=799  C=800  C=801  C=802  C=803  C=804  C=805  C=806  C=807  C=808  C=809  C=810  C=811  C=812  C=813  C=814  C=815  C=816  C=817  C=818  C=819  C=820  C=821  C=822  C=823  C=824  C=825  C=826  C=827  C=828  C=829  C=830  C=831  C=832  C=833  C=834  C=835  C=836  C=837  C=838  C=839  C=840  C=841  C=842  C=843  C=844  C=845  C=846  C=847  C=848  C=849  C=850  C=851  C=852  C=853  C=854  C=855  C=856  C=857  C=858  C=859  C=860  C=861  C=862  C=863  C=864  C=865  C=866  C=867  C=868  C=869  C=870  C=871  C=872  C=873  C=874  C=875  C=876  C=877  C=878  C=879  C=880  C=881  C=882  C=883  C=884  C=885  C=886  C=887  C=888  C=889  C=890  C=891  C=892  C=893  C=894  C=895  C=896  C=897  C=898  C=899  C=900  C=901  C=902  C=903  C=904  C=905  C=906  C=907  C=908  C=909  C=910  C=911  C=912  C=913  C=914  C=915  C=916  C=917  C=918  C=919  C=920  C=921  C=922  C=923  C=924  C=925  C=926  C=927  C=928  C=929  C=930  C=931  C=932  C=933  C=934  C=935  C=936  C=937  C=938  C=939  C=940  C=941  C=942  C=943  C=944  C=945  C=946  C=947  C=948  C=949  C=950  C=951  C=952  C=953  C=954  C=955  C=956  C=957  C=958  C=959  C=960  C=961  C=962  C=963  C=964  C=965  C=966  C=967  C=968  C=969  C=970  C=971  C=972  C=973  C=974  C=975  C=976  C=977  C=978  C=979  C=980  C=981  C=982  C=983  C=984  C=985  C=986  C=987  C=988  C=989  C=990  C=991  C=992  C=993  C=994  C=995  C=996  C=997  C=998  C=999  C=1000  C=1001  C=1002  C=1003  C=1004  C=1005  C=1006  C=1007  C=1008  C=1009  C=1010  C=1011  C=1012  C=1013  C=1014  C=1015  C=1016  C=1017  C=1018  C=1019  C=1020  C=1021  C=1022  C=1023  C=1024  C=1025  C=1026  C=1027  C=1028  C=1029  C=1030  C=1031  C=1032  C=1033  C=1034  C=1035  C=1036  C=1037  C=1038  C=1039  C=1040  C=1041  C=1042  C=1043  C=1044  C=1045  C=1046  C=1047  C=1048  C=1049  C=1050  C=1051  C=1052  C=1053  C=1054  C=1055  C=1056  C=1057  C=1058  C=1059  C=1060  C=1061  C=1062  C=1063  C=1064  C=1065  C=1066  C=1067  C=1068  C=1069  C=1070  C=1071  C=1072  C=1073  C=1074  C=1075  C=1076  C=1077  C=1078  C=1079  C=1080  C=1081  C=1082  C=1083  C=1084  C=1085  C=1086  C=1087  C=1088  C=1089  C=1090  C=1091  C=1092  C=1093  C=1094  C=1095  C=1096  C=1097  C=1098  C=1099  C=1100  C=1101  C=1102  C=1103  C=1104  C=1105  C=1106  C=1107  C=1108  C=1109  C=1110  C=1111  C=1112  C=1113  C=1114  C=1115  C=1116  C=1117  C=1118  C=1119  C=1120  C=1121  C=1122  C=1123  C=1124  C=1125  C=1126  C=1127  C=1128  C=1129  C=1130  C=1131  C=1132  C=1133  C=1134  C=1135  C=1136  C=1137  C=1138  C=1139  C=1140  C=1141  C=1142  C=1143  C=1144  C=1145  C=1146  C=1147  C=1148  C=1149  C=1150  C=1151  C=1152  C=1153  C=1154  C=1155  C=1156  C=1157  C=1158  C=1159  C=1160  C=1161  C=1162  C=1163  C=1164  C=1165  C=1166  C=1167  C=1168  C=1169  C=1170  C=1171  C=1172  C=1173  C=1174  C=1175  C=1176  C=1177  C=1178  C=1179  C=1180  C=1181  C=1182  C=1183  C=1184  C=1185  C=1186  C=1187  C=1188  C=1189  C=1190  C=1191  C=1192  C=1193  C=1194  C=1195  C=1196  C=1197  C=1198  C=1199  C=1200  C=1201  C=1202  C=1203  C=1204  C=1205  C=1206  C=1207  C=1208  C=1209  C=1210  C=1211  C=1212  C=1213  C=1214  C=1215  C=1216  C=1217  C=1218  C=1219  C=1220  C=1221  C=1222  C=1223  C=1224  C=1225  C=1226  C=1227  C=1228  C=1229  C=1230  C=1231  C=1232  C=1233  C=1234  C=1235  C=1236  C=1237  C=1238  C=1239  C=1240  C=1241  C=1242  C=1243  C=1244  C=1245  C=1246  C=1247  C=1248  C=1249  C=1250  C=1251  C=1252  C=1253  C=1254  C=1255  C=1256  C=1257  C=1258  C=1259  C=1260  C=1261  C=1262  C=1263  C=1264  C=1265  C=1266  C=1267  C=1268  C=1269  C=1270  C=1271  C=1272  C=1273  C=1274  C=1275  C=1276  C=1277  C=1278  C=1279  C=1280  C=1281  C=1282  C=1283  C=1284  C=1285  C=1286  C=1287  C=1288  C=1289  C=1290  C=1291  C=1292  C=1293  C=1294  C=1295  C=1296  C=1297  C=1298  C=1299  C=1300  C=1301  C=1302  C=1303  C=1304  C=1305  C=1306  C=1307  C=1308  C=1309  C=1310  C=1311  C=1312  C=1313  C=1314  C=1315  C=1316  C=1317  C=1318  C=1319  C=1320  C=1321  C=1322  C=1323  C=1324  C=1325  C=1326  C=1327  C=1328  C=1329  C=1330  C=1331  C=1332  C=1333  C=1334  C=1335  C=1336  C=1337  C=1338  C=1339  C=1340  C=1341  C=1342  C=1343  C=1344  C=1345  C=1346  C=1347  C=1348  C=1349  C=1350  C=1351  C=1352  C=1353  C=1354  C=1355  C=1356  C=1357  C=1358  C=1359  C=1360  C=1361  C=1362  C=1363  C=1364  C=1365  C=1366  C=1367  C=1368  C=1369  C=1370  C=1371  C=1372  C=1373  C=1374  C=1375  C=1376  C=1377  C=1378  C=1379  C=1380  C=1381  C=1382  C=1383  C=1384  C=1385  C=1386  C=1387  C=1388  C=1389  C=1390  C=1391  C=1392  C=1393  C=1394  C=1395  C=1396  C=1397  C=1398  C=1399  C=1400  C=1401  C=1402  C=1403  C=1404  C=1405  C=1406  C=1407  C=1408  C=1409  C=1410  C=1411  C=1412  C=1413  C=1414  C=1415  C=1416  C=1417  C=1418  C=1419  C=1420  C=1421  C=1422  C=1423  C=1424  C=1425  C=1426  C=1427  C=1428  C=1429  C=1430  C=1431  C=1432  C=1433  C=1434  C=1435  C=1436  C=1437  C=1438  C=1439  C=1440  C=1441  C=1442  C=1443  C=1444  C=1445  C=1446  C=1447  C=1448  C=1449  C=1450  C=1451  C=1452  C=1453  C=1454  C=1455  C=1456  C=1457  C=1458  C=1459  C=1460  C=1461  C=1462  C=1463  C=1464  C=1465  C=1466  C=1467  C=1468  C=1469  C=1470  C=1471  C=1472  C=1473  C=1474  C=1475  C=1476  C=1477  C=1478  C=1479  C=1480  C=1481  C=1482  C=1483  C=1484  C=1485  C=1486  C=1487  C=1488  C=1489  C=1490  C=1491  C=1492  C=1493  C=1494  C=1495  C=1496  C=1497  C=1498  C=1499  C=1500  C=1501  C=1502  C=1503  C=1504  C=1505  C=1506  C=1507  C=1508  C=1509  C=1510  C=1511  C=1512  C=1513  C=1514  C=1515  C=1516  C=1517  C=1518  C=1519  C=1520  C=1521  C=1522  C=1523  C=1524  C=1525  C=1526  C=1527  C=1528  C=1529  C=1530  C=1531  C=1532  C=1533  C=1534  C=1535  C=1536  C=1537  C=1538  C=1539  C=1540  C=1541  C=1542  C=1543  C=1544  C=1545  C=1546  C=1547  C=1548  C=1549  C=1550  C=1551  C=1552  C=1553  C=1554  C=1555  C=1556  C=1557  C=1558  C=1559  C=1560  C=1561  C=1562  C=1563  C=1564  C=1565  C=1566  C=1567  C=1568  C=1569  C=1570  C=1571  C=1572  C=1573  C=1574  C=1575  C=1576  C=1577  C=1578  C=1579  C=1580  C=1581  C=1582  C=1583  C=1584  C=1585  C=1586  C=1587  C=1588  C=1589  C=1590  C=1591  C=1592  C=1593  C=1594  C=1595  C=1596  C=1597  C=1598  C=1599  C=1600  C=1601  C=1602  C=1603  C=1604  C=1605  C=1606  C=1607  C=1608  C=1609  C=1610  C=1611  C=1612  C=1613  C=1614  C=1615  C=1616  C=1617  C=1618  C=1619  C=1620  C=1621  C=1622  C=1623  C=1624  C=1625  C=1626  C=1627  C=1628  C=1629  C=1630  C=1631  C=1632  C=1633  C=1634  C=1635  C=1636  C=1637  C=1638  C=1639  C=1640  C=1641  C=1642  C=1643  C=1644  C=1645  C=1646  C=1647  C=1648  C=1649  C=1650  C=1651  C=1652  C=1653  C=1654  C=1655  C=1656  C=1657  C=1658  C=1659  C=1660  C=1661  C=1662  C=1663  C=1664  C=1665  C=1666  C=1667  C=1668  C=1669  C=1670  C=1671  C=1672  C=1673  C=1674  C=1675  C=1676  C=1677  C=1678  C=1679  C=1680  C=1681  C=1682  C=1683  C=1684  C=1685  C=1686  C=1687  C=1688  C=1689  C=1690  C=1691  C=1692  C=1693  C=1694  C=1695  C=1696  C=1697  C=1698  C=1699  C=1700  C=1701  C=1702 
```


extraction of adverse drug events from biomedical text," in Proceedings of the 2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM 2014), Belfast, UK, Mar. 2014, pp. 54-59.

5. G. Zhu, Z. Sun, S. Zhang, S. Wei, and K. Li, "Causality Extraction Model Based on Two-stage GCN," in Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN 2022), April 2022, pp. 1-8.

6. Q. Li, J. Li, J. Sheng, S. Cui, J. Wu, Y. Hei, H. Peng, S. Guo, L. Wang, A. Beheshti, and P. S. Yu, "A Survey on Deep Learning Event Extraction: Approaches and Applications," in Proceedings of the 2022 IEEE International Conference on Data Mining (ICDM 2022), Dec. 2022, pp. 1-10.

7. A. Akkasi and M. Moens, "Causal relationship extraction from biomedical text using deep neural models: A comprehensive survey," *Journal of Biomedical Informatics*, vol. 117, June 2021, p. 103729.

8. R. Ortega, A. Fonseca, Y. Gutierrez, and A. Montoyo, "SSA-UO: Unsupervised Twitter Sentiment Analysis," in Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis (WASSA 2013), June 2013, pp. 122-127.

9. M.A. Valles Coral, L. Salazar-Ramírez, R. Injante, E.A. Hernandez-Torres, J. JuárezDíaz, J.R. Navarro Cabrera, L. Pinedo, and P. Vidaurre-Rojas, "Density-Based Unsupervised Learning Algorithm to Categorize College Students into Dropout Risk Levels," in Proceedings of the IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2022, pp. 1-6.

10. Sharma, R., Palshikar, G., & Pawar, S. (2018). An Unsupervised Approach for CauseEffect Relation Extraction from Biomedical Text. In 2018 IEEE International Conference on Data Mining Workshops (ICDMW) (pp. 1118-1125). IEEE.

11. Yang, J., Han, S.C. & Poon, J. A survey on extraction of causal relations from natural language text. *Knowl Inf Syst* 64, 1161–1186 (2022).

12. Tirthankar Dasgupta, Rupsa Saha, Lipika Dey, and Abir Naskar. 2018. Automatic Extraction of Causal Relations from Text using Linguistically Informed Deep Neural Networks. In Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue, pages 306–316, Melbourne, Australia. Association for Computational Linguistics.

13. Q. Li et al., "A Survey on Deep Learning Event Extraction: Approaches and Applications," in *IEEE Transactions on Neural Networks and*

Learning Systems, 2022, doi: 10.1109/TNNLS.2022.3213168.

14. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

15. Edoardo Maria Ponti and Anna Korhonen. 2017. Event-Related Features in Feedforward Neural Networks Contribute to Identifying Causal Relations in Discourse. In Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics, pages 25–30, Valencia, Spain. Association for Computational Linguistics.

16. Khoo, C.S., Kornfilt, J., Oddy, R.N., Myaeng, S.H.: Automatic extraction of cause-effect information from newspaper text without knowledge-based inferencing. *Lit. Linguist. Comput.* 13(4), 177–186 (1998)

17. Sorgente A, Vettigli G, Mele F (2013) Automatic extraction of cause-effect relations in natural language text. *DART@ AI* IA* 1109:37–48

18. Kang N, Singh B, Bui QC, Afzal Z, van Mulligen EM, Kors J (2014) Knowledge-based extraction of adverse drug events from biomedical text. *BMC Bioinform* 15:64.



CERTIFICATE