General Language Understanding Evaluation (GLUE) Benchmark

The **General Language Understanding Evaluation (GLUE)** benchmark is a collection of resources designed to train, evaluate, and analyze natural language understanding (NLU) systems.

The GLUE benchmark consists of nine diverse tasks designed to evaluate different aspects of natural language understanding. Here are the tasks:

1. **CoLA (Corpus of Linguistic Acceptability)**: Determines whether a sentence is grammatically correct.
2. **SST-2 (Stanford Sentiment Treebank)**: Classifies the sentiment of a sentence as positive or negative.
3. **MRPC (Microsoft Research Paraphrase Corpus)**: Identifies if two sentences are paraphrases of each other.
4. **STS-B (Semantic Textual Similarity Benchmark)**: Measures the semantic similarity between two sentences.
5. **QQP (Quora Question Pairs)**: Detects if a pair of questions are semantically equivalent.
6. **MNLI (Multi-Genre Natural Language Inference)**: Involves determining the relationship between a premise and a hypothesis, with matched (MNLI-m) and mismatched (MNLI-mm) genres.
7. **QNLI (Question Natural Language Inference)**: Converts the Stanford Question Answering Dataset (SQuAD) into a binary classification task.

Why GLUE (General Language Understanding Evaluation) is Important

GLUE plays a crucial role in advancing the field of NLP and machine learning. It provides a standardized framework for evaluating and comparing different language models, allowing researchers and developers to assess the progress in language understanding algorithms. By setting a common benchmark, GLUE encourages the development of more effective and generalizable language models that can handle a wide range of NLP tasks. It fosters collaboration, promotes transparency, and helps drive innovation in the field.

The Most Important GLUE (General Language Understanding Evaluation) Use Cases

GLUE has various applications in the field of NLP and machine learning. Some of the important use cases include:

* Sentiment Analysis: Assessing the sentiment of a given text, such as determining whether a customer review is positive or negative.
* Text Classification: Categorizing text into predefined classes or categories based on its content.
* Named Entity Recognition: Identifying and classifying named entities in text, such as person names, organizations, and locations.
* Text Similarity: Measuring the similarity between two pieces of text, which has applications in information retrieval and recommendation systems.
* Question Answering: Automatically finding relevant answers to user questions based on a given context or a set of documents.

Named entity Recognition

**Named Entity Recognition (NER)** is a technique in **natural language processing (NLP)** that focuses on identifying and classifying entities.

The purpose of NER is to automatically extract **structured information from unstructured text,** enabling machines to understand and categorize entities in a meaningful manner for various applications like text summarization, building knowledge graphs, question answering, and knowledge graph construction.

**NLP is just a two-step process**, below are the two steps that are involved:

* Detecting the entities from the text
* Classifying them into different categories

How to Implement NER in Python?

For implementing NER system, we will leverage Spacy library. The code can be run on colab, however for visualization purpose. I recommend the local environment. We can install the required libraries using:

* !pip install spacy
* !pip install nltk
* ! python -m spacy download en\_core\_web\_sm

import pandas as pd

import spacy

import requests

from bs4 import BeautifulSoup

nlp = spacy.load("en\_core\_web\_sm")

pd.set\_option("display.max\_rows", 200)

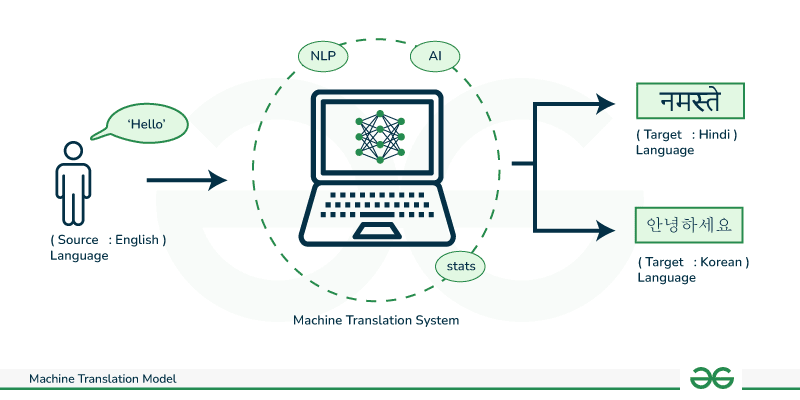
[SpaCy,](https://www.geeksforgeeks.org/python-named-entity-recognition-ner-using-spacy/) a natural language processing library to process text and extract named entities. The code iterates through the named entities identified in the processed document and printing each entity’s text, start character, end character and label.

Refer the code in github NER

Machine Translation

**Machine translation** of languages refers to the use of **artificial intelligence (AI)** and machine learning algorithms to automatically translate text or speech from one language to another.

 In [Natural Language Processing (NLP)](https://www.geeksforgeeks.org/natural-language-processing-overview/), the goal of machine translation is to produce translations that are not only grammatically correct but also convey the meaning of the original content accurately.



**What are the key approaches in Machine Translation?**

In machine translation, the original text is decoded and then encoded into the target language through two step process that involves various approaches employed by language translation technology to facilitate the translation mechanism.

**1. Rule-Based Machine Translation**

Rule-based machine translation relies on these resources to ensure precise translation of specific content. The process involves the software parsing input text, generating a transitional representation, and then converting it into the target language **with reference to grammar rules and dictionaries.**

**2. Statistical Machine Translation**

Rather than depending on linguistic rules, [statistical machine translation](https://www.geeksforgeeks.org/statistical-machine-translation-of-languages-in-artificial-intelligence/) utilizes machine learning for text translation. Machine learning algorithms examine extensive human translations, identifying statistical patterns. When tasked with translating a new source text, the software intelligently guesses based on the statistical likelihood of specific words or phrases being associated with others in the target language.

**3. Neural Machine Translation (NMT)**

A [neural network](https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/), inspired by the human brain, is a network of interconnected nodes functioning as an information system. Input data passes through these nodes to produce an output. Neural machine translation software utilizes neural networks to process vast datasets, with each node contributing a specific change from source text to target text until the final result is obtained at the output node.

**4. Hybrid Machine Translation**

Hybrid machine translation tools integrate multiple machine translation models within a single software application, leveraging a combination of approaches to enhance the overall effectiveness of a singular translation model. This process typically involves the incorporation of rule-based and statistical machine translation subsystems, with the ultimate translation output being a synthesis of the results generated by each subsystem.

**Why we need Machine Translation in NLP?**

1.Cross-border communication

2.Localization

3.Business

4.Education

5.Government

Natural Language Understanding (NLU)

Natural Language Understanding (NLU) is the ability of a computer to understand human language. You can use it for many applications, such as chatbots, voice assistants, and automated translation services.

The most basic form of NLU is parsing, which takes text written in natural language and converts it into a structured format that computers can understand.

For example, the words "hello world" would be converted into their respective parts of speech (nouns and verbs), while "I am hungry" would be split into two sentences: "I am" and "hungry."

Parsing is only one part of NLU; other tasks include sentiment analysis, entity recognition, and semantic role labeling.

**NLU vs. NLP vs. NLG**

Natural language understanding is taking in an input text string and analyzing what it means.

N[atural language processing](https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/what-is-natural-language-processing-nlp) is the process of turning human-readable text into computer-readable data. It's used in everything from online search engines to chatbots that can understand our questions and give us answers based on what we've typed.

Natural language generation is the process of turning computer-readable data into human-readable text.

Text Summarization

Automatic text summarization refers to a group of methods that employ algorithms to compress a certain amount of text while preserving the text’s key points.

**Types of Text Summarization**

There are typically two basic methods for automatic text summarization:

1. Extractive summarization
2. Abstractive summarization

Extractive summarization algorithms are employed to generate a summary by selecting and combining key passages from the source material. Unlike humans, these models emphasize creating the most essential sentences from the original text rather than generating new ones. Extractive summarization utilizes the [Text Rank algorithm](https://www.geeksforgeeks.org/page-rank-algorithm-implementation/)

**Prerequisite**

**Spacy**

To Install the Spacy and Dowload the English Language Dependency run the below code in terminal

!pip install spacy

To install the english laguage dependency

!python3 -m spacy download en\_core\_web\_lg

**TextRank**

To Install the TextRank

!pip install pytextrank

**Abstractive Summarization:**

Abstractive summarization techniques emulate human writing by generating entirely new sentences to convey key concepts from the source text, rather than merely rephrasing portions of it. These fresh sentences distill the vital information while eliminating irrelevant details, often incorporating novel vocabulary absent in the original text. The term “Transformers” has recently dominated the natural language processing field, although these models initially relied on designs based on recurrent[neural networks](https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/) (RNNs).

**What Are Transformers?**

[Transformers](https://www.geeksforgeeks.org/what-is-the-difference-between-transform-and-fit_transform-in-sklearn-python/) represent a series of systems that employ a unique encoder-decoder architecture to transform an input sequence into an output sequence.

Transfer Learning with Neural Networks

Using pre-trained models and model weights

A neural network is trained on a data. This network gains knowledge from this data, which is compiled as “weights” of the network. These weights can be extracted and then transferred to any other neural network. Instead of training the other neural network from scratch, we **“transfer”** the learned features.

What is a Pre-trained Model?

a pre-trained model is a model created by some one else to solve a similar problem. Instead of building a model from scratch to solve a similar problem, you use the model trained on other problem as a starting point.

Some popular pre-trained models for NLP include BERT, GPT-2, ELMo, and RoBERTa. These models are trained on large datasets of text and can be fine-tuned for specific tasks.

**Why do we use Pretrained Models?**

* Reduces the computational burden required for initial model training hence, making development more accessible.
* The learned knowledge can be used for various applications.
* Models can be fine-tuned according to the task and can result in superior performance to training from the initial point.
* Less labelled data is required for fine-tuning specific tasks.

The application of pretrained models is not limited to NLP, it is also used for[image classification](https://www.geeksforgeeks.org/multiclass-image-classification-using-transfer-learning/), [image segmentation](https://www.geeksforgeeks.org/image-segmentation-using-k-means-clustering/) and other [computer vision](https://www.geeksforgeeks.org/computer-vision/) applications.

Techniques for fine-tuning and adapting models to new tasks

Fine-tuning and adapting models to new tasks is a crucial aspect of machine learning, especially when leveraging pre-trained models. Here are some key techniques:

**1. Hyperparameter Tuning**

Adjusting hyperparameters such as learning rate, batch size, and the number of epochs can significantly impact model performance. [Fine-tuning these parameters helps optimize the model for the new task](https://www.markovml.com/blog/model-fine-tuning).

**2. Transfer Learning**

This involves using a pre-trained model and adapting it to a new task. [Typically, the initial layers are frozen to retain learned features, while the later layers are retrained on the new dataset](https://www.ibm.com/topics/fine-tuning).

**3. Data Augmentation**

[Creating modified versions of the training data (e.g., rotations, flips, and color changes) can help increase the dataset’s diversity and reduce overfitting](https://www.markovml.com/blog/model-fine-tuning).

**4. Regularization Techniques**

[Methods like dropout, L2 regularization, and early stopping can prevent overfitting by adding constraints to the model’s complexity](https://www.markovml.com/blog/model-fine-tuning).

**5. Top-Layer Tuning**

Initially, only the top layers of the model are retrained on the new data. [This approach allows the model to adapt to the new task without altering the core features learned by the lower layers3](https://www.multimodal.dev/post/understanding-fine-tuning-in-deep-learning).

**6. Learning Rate Adjustments**

[Starting with a lower learning rate can help fine-tune the model more delicately, preventing drastic changes to the pre-trained weights](https://www.markovml.com/blog/model-fine-tuning)[3](https://www.multimodal.dev/post/understanding-fine-tuning-in-deep-learning).

**7. Low-Rank Adaptation (LoRA)**

[Techniques like LoRA and QLoRA (Quantized Low-Rank Adaptation) reduce the number of trainable parameters, making the fine-tuning process more efficient and preventing catastrophic forgetting](https://www.markovml.com/blog/model-fine-tuning)[4](https://www.acorn.io/resources/learning-center/fine-tuning-llm).

Frozen layers and trainable layers

**Frozen Layers**

* **Definition**: Frozen layers are layers whose weights are not updated during training. This means they retain the knowledge they were initially trained with.
* **Purpose**: Freezing layers is useful when you want to leverage pre-trained models. By freezing the initial layers, you preserve the learned features and avoid altering them during further training.
* **How to Freeze**: In frameworks like TensorFlow and Keras, you can set the trainable attribute of a layer to False to freeze it. For example:

for layer in model.layers:

layer.trainable = False

**Trainable Layers**

* **Definition**: Trainable layers are layers whose weights are updated during training. These layers learn and adapt to the new data provided.
* **Purpose**: Adding trainable layers on top of frozen layers allows the model to learn new features specific to the new task while retaining the general features learned from the pre-trained model.
* **How to Set Trainable**: By default, layers are trainable. You can explicitly set a layer to be trainable by setting the trainable attribute to True:

for layer in model.layers:

layer.trainable = True

**Typical Workflow in Transfer Learning**

1. **Load a Pre-trained Model**: Start with a model pre-trained on a large dataset (e.g., ImageNet).
2. **Freeze Initial Layers**: Freeze the layers of the pre-trained model to retain their learned features.
3. **Add New Trainable Layers**: Add new layers on top of the frozen layers to adapt the model to the new task.
4. **Train the New Layers**: Train the new layers on your specific dataset.
5. [**Fine-tuning (Optional)**: Unfreeze some of the initial layers and train the entire model with a low learning rate to fine-tune the pre-trained features to the new data](https://www.tensorflow.org/guide/keras/transfer_learning)

[transfer\_learning - Colab (google.com)](https://colab.research.google.com/github/keras-team/keras-io/blob/master/guides/ipynb/transfer_learning.ipynb)